Lecture Notes
Computer Sc

Kyung-Yong Chwa

Algorithms
and Compu

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Kyung-Yong Chwa   Oscar H. Ibarra (Eds.)

Algorithms and Computation

9th International Symposium, ISAAC'98
Taejon, Korea, December 14-16, 1998
Proceedings

Springer

Volume Editors

Kyung-Yong Chwa
Korea Advanced Institute of Science and Technology
Department of Computer Science
373-1, Kusong-dong, Yusong-gu, Taejon 305-701, Korea
E-mail: kychwa@jupiter.kaist.ac.kr

Oscar H. Ibarra
University of California, Department of Computer Science
Santa Barbara, CA 93106, USA
E-mail: ibarra@cs.ucsb.edu

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Algorithms and computation : 9th international symposium ; proceedings / ISAAC '98, Taejon, Korea, December 14 -16, 1998. Kyung-Yong Chwa ; Oscar H. Ibarra (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1998
(Lecture notes in computer science ; Vol. 1533)
ISBN 3-540-65385-6

CR Subject Classification (1998): F.2, F.1, C.2, G.2-3, I.3.5

ISSN 0302-9743
ISBN 3-540-65385-6 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10692948 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The papers in this volume were selected for presentation at the Ninth Annual International Symposium on Algorithms and Computation (ISAAC'98), held on December 14–16, 1998 in Taejon, Korea. Previous meetings were held in Tokyo (1990), Taipei (1991), Nagoya (1992), Hong Kong (1993), Beijing (1994), Cairns (1995), Osaka (1996), and Singapore (1997).

The symposium was jointly sponsored by Korea Advanced Institute of Science and Technology (KAIST) and Korea Information Science Society (KISS) to commemorate its 25th anniversary in cooperation with Ministry of Information and Communication, Korea Information Society Development Institute, and Korea Science and Engineering Foundation.

In response to the call for papers, 102 extended abstracts were submitted from 21 countries. Each submitted paper was reported on by at least four program committee members, with the assistance of referees, as indicated by the referee list found in these proceedings. There were many more acceptable papers than there was space available in the symposium schedule, and the program committee's task was extremely difficult. The 47 papers selected for presentation had a total of 105 authors, resident as follows: Japan 24, Germany 17, United State of America 15, Taiwan 10, Hong Kong and Korea 6 each, Spain 5, Switzerland and Australia 4 each, Austria, Canada, and France 3 each, Italy and Netherlands 2 each, and Greece 1.

We thank all program committee members and their referees for their excellent work, especially given the demanding time constraints; they gave the symposium its distinctive character. We thank all who submitted papers for consideration: they all contributed to the high quality of the symposium.

Finally, we thank all the people who worked hard to put in place the logistical arrangements of the symposium — our colleagues and our graduate students. It is their hard work that made the symposium possible and enjoyable.

September 1998

Kyung-Yong Chwa
Oscar Ibara

Conference Organization

Program Committee Co-Chairs	Kyung-Yong Chwa (KAIST, Korea) Oscar Ibara (UC Santa Barbara, USA)
Program Committee	Takao Asano (Chuo U., Japan) Ding-Zhu Du (U. of Minnesota, USA) Susanne Hambrusch (Purdue U., USA) Hiroshi Imai (U. of Tokyo, Japan) Tao Jiang (McMaster U., Canada) Sam Kim (Kyungpook Nat. U., Korea) D.T. Lee (Northwestern U., USA) Ming Li (U. of Waterloo, Canada) Pandu Rangan (IIT, Madras, India) Sartaj Sahni (U. of Florida, USA) P. Spirakis (Comp. Tech. Inst., Patras, Greece) Roberto Tamassia (Brown U., USA) Shanghua Teng (U. of Illinois, USA) Osamu Watanabe (Tokyo Inst. Tech., Japan) Peter Widmayer (ETH, Zürich, Switzerland) Chee K. Yap (Courant Inst., NYU, USA) Hsu-Chun Yen (National Taiwan U., Taiwan)
Organizing Committee Chair	Jik Hyun Chang (Sogang U., Korea)
Organizing Committee	Hee-Chul Kim (Hankuk U. of Foreign Studies, Korea) Sang-Ho Lee (Ewha Womans U., Korea) Kunsoo Park (Seoul Nat. U., Korea)

Sponsors

Ministry of Information and Communication
Korea Advanced Institute of Science and Technology
Korea Information Science Society
Information Processing Society of Japan
Korea Information Society Development Institute
Korea Science and Engineering Foundation

Referees

Alender, Eric
Bai, Mino
Bellare, Mihir
Bultan, Tevfik
Chang, Jik Hyun
Chang, Shi-Chung
Chen, Donghui
Chen, Ming-Syan
Cheng, Xiuzhen
Cho, Hwan-Gue
Cho, Jun-Dong
Cho, Yookun
Chung, Choo Hee
Cole, Richard
Cramer, Ronald
Das, Sajal
de Wolf, Ronald
Domingo, Carlos
Du, David
Egecioglu, Omer
Eidenbenz, Stephan
Farach-Colton, Martin
Fatourou, Panagiota
Fotakis, Dimitris
Galli, Nicola
Garay, Juan
Hahn, Sang Geun
Han, Shitou
Hirata, Tomio
Hsu, Tsan-sheng
Igarashi, Yoshihide
Imai, Keiko
Inaba, Mary
Janicki, Ryszard
Kaklamanis, Christos
Kaneko, Taishou
Kim, Chul E.
Kim, Dongsoo
Kim, Sang Dong
Kim, Sung Ho
Kim, Sung Kwon
Kim, Yeong-Dae
Ko, Ming-Tat
Konheim, Alan
Kontogiannis, Spyros
Koshiba, Takeshi
Krithivasan, Kamala
Kurosawa, Kaoru
Lanctot, Kevin
Lee, Dong Hoon
Lee, Jae-Ha
Lee, Sang-Ho
Lee, Seungyong
Lei, Chin-Laung
Lim, Hyung-Seok
Lin, Mao-Chao
Liotta, Giuseppe
Lu, Bing
Lyuu, Yuh-Dauh
Ma, Bin
Ma, Tse-Heng
Morita, Ken'ichi
Nakano, Koji
Nakano, Shin-ichi
Neyer, Gabriele
Ogata, Wakaha
Okamoto, Ryumei
Olariu, Stephan
Pajarola, Renato
Palis, Michael
Pardalos, Panos
Park, Chong-Dae
Park, Jung-Heum
Park, Kunsoo
Rajasekaran, Sanguthevar
Rim, Chong S.
Roos, Thomas
Ruan, Lu
Ryu, Kwan Woo
Schlude, Konrad
Shin, Chan-Su
Singh, Ambuj
Smith, Terence
Stamm, Christoph
Steiner, George
Sung, Ting-Yi
Suzuki, Hitoshi
Tang, Chuan-Yi
Toda, Seinosuke
Tokuyama, Takeshi
Trachsler, Beat
Tromp, John
Tsai, Shi-Chun
Tsay, Yih-Kuen
Uehara, Ryuhei
Ulber, Roland
Verbeurgt, Karsten
Vitányi, Paul
Wagner, Dorothea
Wang, Da-Wei
Wang, Yue-Li
Wareham, Todd
Watanabe, Toshimasa
Wattenhofer, Roger
Wolf, Stefan
Wu, Xiaoyu
Yamashita, Masafumi
Yamazaki, Koichi
Yang, Tae-Cheon
Yang, Tao
Yang, Wuu
Yoo, Kee Yung
Yoo, Kwan-Hee
Yu, Sheng
Zhang, Louxin
Zhou, Xiao

Table of Contents

The Discrepancy Method[*]
(Invited Presentation)

Bernard Chazelle

Princeton University, USA
and
Ecole Polytechnique, France

Abstract. Discrepancy theory is the study of irregularities of distributions. A typical question is: given a "complicated" distribution, find a "simple" one that approximates it well. As it turns out, many questions in complexity theory can be reduced to problems of that type. This raises the possibility that the deep mathematical techniques of discrepancy theory might be of utility to theoretical computer scientists. As will be discussed in this talk this is, indeed, the case. We will give several examples of breakthroughs derived through the application of the "discrepancy method."

In 1935, van der Corput [21,22] asked how uniform an infinite sequence of numbers in $[0,1]$ can be. Specifically, how slowly does the function

$$D(n) = \sup_{0 \le x \le 1} \Big| \, |S_n \cap [0,x]\,| - nx \, \Big|$$

grow with n, where S_n consists of the first n elements in the sequence? A surprising theorem of Schmidt says that $D(n)$ can never be in $o(\log n)$.

Schmidt's result sets a limit on how well a certain discrete distribution, $x \mapsto |S_n \cap [0,x]\,|$, can simulate a continuous one, $x \mapsto nx$: thus, a certain amount of *discrepancy* between the two distributions is unavoidable. A computer scientist reading this paragraph should go back to the previous sentence and replace the words "discrete" by "easy to compute" and "continuous" by "hard to compute." The resulting sentence talks about efficiently computable distributions simulating intractable ones, which happens to be the driving issue behind the use of randomization in computer science.

A probabilistic algorithm, by its very nature, requires access to a sequence of perfectly random bits. For all sorts of theoretical reasons there is considerable interest in finding ways to reduce the amount of randomness, ie, to get by with pseudorandom sequences, or even better, to eliminate randomness altogether (derandomization).

This is where discrepancy theory comes into play. The theory is blessed with many powerful tools and techniques developed over the last century. The

[*] This work was supported in part by NSF Grant CCR-93-01254, NSF Grant CCR-96-23768, ARO Grant DAAH04-96-1-0181, and NEC Research Institute.

discrepancy method bridges these tools with complexity theory and algorithm design. It has been the force behind major recent developments in areas as diverse as probabilistic algorithms, derandomization, communication complexity, searching, machine learning, pseudo-randomness, computational geometry, optimization, and computer graphics. The talk briefly samples some of these achievements: in particular, optimal algorithms for convex hull and Voronoi diagrams [2], lower bounds for arithmetic circuits [4] and range searching [3], deterministic linear-time algorithms for linear programming with a constant number of variables [6,8,15], as well as results in pseudorandomness [9,10,12,18,19], communication complexity [11], derandomization [13,17]. For references on discrepancy theory in general, see [1,14,16,7,20], and on the discrepancy method in particular, see [5].

References

1. Beck, J., Chen, W.W.L. *Irregularities of distribution*, Cambridge Tracts in Mathematics, 89, Cambridge Univ. Press, Cambridge, 1987. 2
2. Chazelle, B. *An optimal convex hull algorithm in any fixed dimension*, Disc. Comput. Geom., 10 (1993), 377–409. 2
3. Chazelle, B. *Lower bounds for off-line range searching*, Disc. Comput. Geom., 17 (1997), 53–65. 2
4. Chazelle, B. *A spectral approach to lower bounds with applications to geometric searching*, SIAM J. Comput., 27 (1998), 545–556. 2
5. Chazelle, B. *The Discrepancy Method*, book in preparation, 1998. 2
6. Chazelle, B., Matoušek, J. *On linear-time deterministic algorithms for optimization problems in fixed dimension*, J. Algorithms, 21 (1996), 579–597. 2
7. Drmota, M.,Tichy, R.F., *Sequences, Discrepancies and Applications*, Springer-Verlag, Lecture Notes in Mathematics, Vol. 1651, March 1997. 2
8. Dyer, M.E. *A class of convex programs with applications to computational geometry*, Proc. 8th Annu. ACM Symp. Comput. Geom. (1992), 9–15. 2
9. Even, G., Goldreich, O., Luby, M., Nisan, N., Veličković, B. *Efficient approximation of product distributions*, Random Structures & Algorithms, 13 (1998), 1–16. 2
10. Impagliazzo, R., Zuckerman, D. *How to recycle random bits*, Proc. 30th Annu. IEEE Symp. Found. Comput. Sci. (1989), 248–253. 2
11. Kushilevitz, E., Nisan, N. *Communication Complexity*, Cambridge Univ. Press, 1997 2
12. Luby, M. *Pseudorandomness and Cryptographic Applications*, Princeton Computer Science Notes, Princeton University Press, 1996. 2
13. Matoušek, J. *Derandomization in computational geometry*, J. Algorithms, 20 (1996), 545–580. 2
14. Matoušek, J. *Geometric Discrepancy: An Illustrated Guide*, Springer-Verlag, in press. 2
15. Megiddo, N. *Linear programming in linear time when the dimension is fixed*, J. ACM, 31 (1984), 114–127. 2
16. Niederreiter, H. *Random Number Generation and Quasi-Monte Carlo Methods*, CBMS-NSF, SIAM, Philadelphia, PA, 1992. 2
17. Razborov, A., Szemerédi, Wigderson, A. *Constructing small sets that are uniform in arithmetic progressions*, Combinatorics, Probability and Computing, 2 (1993), 513–518. 2

18. Sinclair, A. *Algorithms for Random Generation and Counting : A Markov Chain Approach*, Progress in Theoretical Computer Science, Birkhauser, 1993. 2
19. Sipser, M. *Expanders, Randomness, or Time versus Space,* Proc. 1st Annu. Conf. Structure in Complexity Theory (1986), 325–329. 2
20. Spencer, J. *Ten Lectures on the Probabilistic Method*, CBMS-NSF, SIAM, 1987. 2
21. van der Corput, J.G. *Verteilungsfunktionen* I. Proc. Nederl. Akad. Wetensch., 38 (1935), 813–821. 1
22. van der Corput, J.G. *Verteilungsfunktionen* II. Proc. Nederl. Akad. Wetensch., 38 (1935), 1058–1066. 1

Implementing Algorithms and Data Structures: An Educational and Research Perspective
(Invited Presentation)

Roberto Tamassia

Department of Computer Science, Brown University
Providence, RI 02912-1910, USA
rt@cs.brown.edu
http://www.cs.brown.edu/people/rt/

Abstract. Anecdotal evidence shows that developing correct and efficient implementations of fundamental data structures and algorithms is a challenging task. We discuss educational and research issues in algorithm engineering. Topics include algorithmic patterns, the development of a library of data structures and algorithms in Java, and the use of design patters in teaching introductory data structures courses.

1 Introduction

Developing correct, efficient, and reusable implementations of fundamental data structures and algorithms is a challenging task (see, e.g., [15]). The LEDA project [11] is major effort in this direction. The emerging research area of *algorithm engineering* investigates techniques for the implementation, testing, and evaluation of algorithms in realistic environments and scenarios.

In this talk, we discuss algorithm engineering issues from an educational and research perspective. We focus on methods for implementing algorithms and data structures so that they are generic and extensible and on the use of software engineering concepts, such as object-oriented programming and design patterns.

2 Algorithmic Patterns

Abstraction has been, in the past years, a key technique for handling the increasing complexity of programs. Algorithmic abstraction is the process aimed at the implementation of sophisticated algorithms as reusable software objects. It goes beyond both procedural abstraction (structured programming) and data abstraction (object-oriented programming), by viewing algorithms as objects that can be manipulated at the programming language level. It allows the programmer to construct new algorithms by specifying modifications and extensions of existing ones, thus reducing the time spent coding and debugging. It allows predefined algorithms to be combined in complex ways. These predefined algorithms of common use, called *algorithmic patterns*, are the main tools for algorithm abstraction. In many cases, algorithmic patterns abstract out the core of several similar algorithms, and the programmer only has to implement the extensions of the core that are needed for a specific algorithm.

In [5], we describe an object-oriented design and implementation of the core steps of the GIOTTO algorithm [13] for constructing orthogonal drawings of graphs. The design is based on a new algorithmic pattern called *algorithmic reduction*, which provides conversion among different types of data structures. GIOTTO is particularly suitable as a case study for the use of algorithmic reductions because it consists of many steps where the original graph is converted into various other structures (e.g., a flow network).

In [14], we present the design and implementation of a new algorithmic pattern, called *binary space partition search*, that provides a unifying description of two well-known algorithms for the fundamental geometric problem of point location in a planar map [12]. We also show how to separate the topological, geometric, and numerical aspects of the computation. Further algorithmic patterns within the area of geometric computing are being investigated, including plane-sweep, lifting map, and fractional cascading.

3 JDSL: A Library of Data Structures in Java

JDSL, the *Data Structure Library in Java*, is a new library of fundamental data structures and algorithms being developed at Brown and at Johns Hopkins.

The architecture of JDSL is partitioned into four different components, namely the *combinatorial*, the *topological*, the *geometric*, and the *arithmetic* components. Each component of JDSL corresponds to one or more Java packages. Each package consists of a collection of interfaces, and for each interface one or more reference implementations are provided. The interfaces are arranged in hierarchies that may extend across different packages or components (see Fig. 1). The design principles of JDSL do not directly depend on Java; their validity extends to C++ and other object-oriented languages.

Several reference implementations provided by JDSL are optimized for fast running time. For example, the *tiered vector* implementation of a sequence [7] is considerably faster than the standard Java vector class. JDSL also allows rapid prototyping of complex algorithms and data structures by means of combinations of algorithmic patterns. This programming style may yield implementations that are slower than ad-hoc ones due to the indirection overhead.

In [6] we present two key concepts used in JDSL: positions and locators. Many applications require data structures that allow efficient access to their internal organization and to their elements. This feature has been implemented in some libraries with *iterators* or *items*. JDSL refines the notion of an item and splits it into two related concepts: *positions* and *locators*. Positions are an abstraction of a pointer to a node or an index into an array; they provide direct access to the in-memory structure of the container. Locators add a level of indirection and

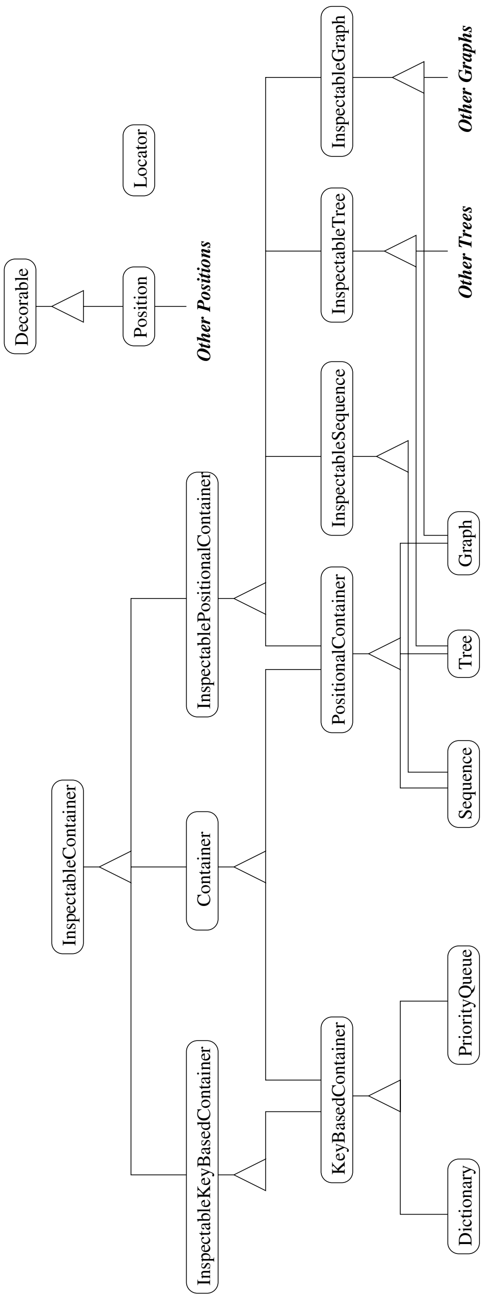


Fig. 1. Partial view of the JDSL interface hierarchy.

allow the user to find a specific element even if the position holding the element changes.

A number of implementation case studies have been conducted based on JDSL, including data structures for planar maps [9], animations of geometric algorithms [2], and the aforementioned work on point location [14] and graph drawing [5]

The JDSL library has a companion *teach* version [10] suited for instructional use. It includes a simplified combinatorial component and provides support for visualizers and testers [1]. The code examples in the data structures textbook by Goodrich and Tamassia [8] are taken from this library. Classroom experience at Brown and Johns Hopkins has shown that the methodology of the library allows students to complete sophisticated course projects in a short amount of time.

4 Teaching Data Structure Design Patterns

Design patterns [3] are organizational concepts developed for designing quality object-oriented software. A design pattern provides a framework for a "typical" software design problem. It is a general template that can be specialized for the specific problem at hand. Design patters have been shown to save development time and yield software that is robust and reusable. Design patterns are important, but probably neglected by most instructors in the introduction to data structures course (CS2) and usually not taught until the software engineering course.

In [4,8], we show how to incorporate design patterns in the teaching of an introductory data structures course (CS2). The specific design patterns we discuss include *adapter*, *template method*, *comparator*, *decorator*, *iterator*, *position*, and *locator*. Incorporating these patterns does not require any major revisions to the CS2 curriculum, for they fit in naturally with the discussions of several components of CS2 (see Table 1).

Design Pattern	**CS2 Topic**
adapter	stacks and queues
template method	tree and graph traversals
comparator	priority queues
decorator	balanced trees, graphs
iterator	sequences, trees, graphs
position	sequences, binary trees, graphs
locator	priority queues, dictionaries

Table 1. Some design patterns and natural places in the CS2 curriculum where they can be introduced.

Especially important is the *template method* pattern, where a class that implements the skeleton of an algorithm is provided, and the steps that will vary in different implementations are delegated to its subclasses. Template methods can be introduced in CS2 during the discussion of tree and graph traversals.

Acknowledgments

I would like to thank Mike Goodrich for his collaboration to the projects mentioned in this paper.

References

1. R. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel. Testers and visualizers for teaching data structures. Manuscript, 1998. 7
2. J. Beall. Shortest path between two points in a polygon. http://www.cs.brown.edu/courses/cs252/projects/jeb/html/cs252proj.html. 7
3. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, Reading, MA, 1995. 7
4. N. Gelfand, M. T. Goodrich, and R. Tamassia. Teaching data structure design patterns. In *Proc. SIGCSE*, 1997. 7
5. N. Gelfand and R. Tamassia. Algorithmic patterns for graph drawing. In *Proc. Graph Drawing '98*. Springer-Verlag, to appear. 5, 7
6. M. T. Goodrich, M. Handy, B. Hudson, and R. Tamassia. Accessing the internal organization of data structures in the JDSL library. Manuscript, 1998. 5
7. M. T. Goodrich and J. G. Kloss. Tiered vector: an efficient dynamic array for JDSL. Manuscript, 1998. 5
8. M. T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java*. Wiley, New York, NY, 1998. 7
9. D. Jackson. The TripartiteEmbeddedPlanarGraph. Manuscript, 1997. 7
10. JDSL teach version home page. http://www.cs.brown.edu/cgc/jdsl. 7
11. K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, New York, 1998. 4
12. F. P. Preparata. Planar point location revisited. *Internat. J. Found. Comput. Sci.*, 1(1):71–86, 1990. 5
13. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988. 5
14. R. Tamassia, L. Vismara, and J. E. Baker. A case study in algorithm engineering for geometric computing. In *Proc. Workshop on Algorithm Engineering*, pages 136–145, 1997. 5, 7
15. K. Weihe. Reuse of algorithms: Still a challenge to object-oriented programming. In *Proc. OOPSLA '97*, pages 34–48, 1997. 4

$L\infty$ Voronoi Diagrams and Applications to VLSI Layout and Manufacturing

Evanthia Papadopoulou

IBM TJ Watson Research Center, Yorktown Heights, NY 10598, USA
evanthia@watson.ibm.com

Abstract. In this paper we address the L_∞ Voronoi diagram of polygonal objects and present applications in VLSI layout and manufacturing. We show that in L_∞ the Voronoi diagram of segments consists only of straight line segments and is thus much simpler to compute than its Euclidean counterpart. Moreover, it has a natural interpretation. In applications where Euclidean precision is not particularly important the L_∞ Voronoi diagram can provide a better alternative. Using the L_∞ Voronoi diagram of polygons we address the problem of calculating the *critical area* for shorts in a VLSI layout. The critical area computation is the main computational problem in VLSI yield prediction.

1 Introduction

The Voronoi diagram of polygonal objects has been given considerable attention because of its numerous applications in diverse areas such as biology, geography, robot motion planning, computer graphics. In this paper we present applications of Voronoi diagrams in VLSI layout and manufacturing. It is well known that the ordinary Voronoi diagram of polygonal objects has linear combinatorial complexity and consists of straight line segments and parabolic arcs. Several efficient algorithms using divide and conquer, plane sweep, or incremental construction are known for its computation (see [2,3] for a survey). However, the existence of parabolic arcs in the diagram makes it hard to compute in practice. In a recent paper [9], a new complexity model for geometric algorithms was introduced, called the *degree* of an algorithm, which characterizes the complexity of the test computations of an algorithm. In the construction of the Voronoi diagram of segments Burnikel showed that the well-known *incircle test* can be solved with degree 40 [4,9]. This is very high for practical problems involving Voronoi diagrams.

In a recent paper Aichholzer and Aurenhammer [1] introduced a new type of skeleton for polygonal objects in the plane called the *straight skeleton.* The straight skeleton consists of angular bisectors between edges and thus consists of straight line segments. The straight skeleton captures the *shape* of the defining elements in a natural manner. Its main advantage over the Voronoi diagram is the elimination of parabolic arcs. However, straight skeletons do not provide the proximity information that Voronoi diagrams do and thus, they do not always provide an alternative solution to Voronoi diagrams.

In applications where Euclidean accuracy is not particularly important a practical solution to the high degree problem of Euclidean Voronoi diagrams of segments may be the use of a different geometry, in particular, the L_∞-metric. The L_∞ Voronoi diagram of polygons (or segments) consists of straight line segments and its combinatorial complexity is similar to the Euclidean case. Moreover, if the input vertices are on rational coordinates the Voronoi vertices are also rational. An intuitive way to view the L_∞ Voronoi diagram of polygonal shapes is to view it as the locus of points corresponding to centers of isothetic squares[1] that touch the boundary in at least two points. In contrast, the Euclidean Voronoi diagram can be regarded as the locus of centers of circles that touch the boundary in at least two points. Note that in the rectilinear case where the edges of the polygons are either horizontal or vertical, the L_∞ Voronoi diagram coincides with the straight-skeleton of [1].

Our motivation for considering L_∞ Voronoi diagrams comes from applications in VLSI layout and manufacturing, as described below. Note that layout shapes have many orthogonal edges but are not necessarily rectilinear. The proximity information preserved in the L_∞ Voronoi diagram can be used to calculate the *critical area*, a measure used in estimating the sensitivity of a VLSI layout to defects during manufacturing. The critical area calculation is the main computational problem in predicting the *yield* of a VLSI chip. Other applications of Voronoi diagrams in extracting from the physical description of a design the equivalent resistance are discussed in [12,16]. The L_∞ version would be as good in this case and much simpler to obtain [15]. Due to the simplicity of L_∞ Voronoi diagrams we expect them to also find applications in other areas. For example, in [17] an automatic generation for finite element meshes for multiply connected planar domains with polygonal boundaries is described. The Euclidean Voronoi diagram of the domain was used as a starting point and could be substituted by the L_∞ version [15] which is easier to compute.

2 L_∞ Voronoi Diagrams

The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is the maximum between the horizontal and the vertical distance between p and q i.e., $d(p,q) = \max\{d_x(p,q), d_y(p,q)\}$ where $d_x(p,q) = |x_p - x_q|$ and $d_y(p,q) = |y_p - y_q|$. The L_∞ distance between a point p and a line l is $d(p,l) = \min\{d(p,q), \forall q \in l\}$. Intuitively, the L_∞ distance between two elements (points or lines) can be defined in terms of the smallest isothetic square touching the two elements. We shall refer to a line of slope (± 1) simply as a *45-degree* line, denoted 45° line.

Consider a point p and the four 45° rays emanating away from p (see Fig. 1). They partition the plane into four quadrants. In each quadrant the L_∞ distance between p and any point q simplifies to either the vertical (q in upper and lower quadrant) or the horizontal distance (q in right and left quadrant) between p and q. Any non-45° line l that does not contain p intersects two 45° rays at

[1] An isothetic square is one whose sides are parallel to the coordinate axes.

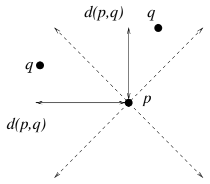


Fig. 1. The 45° rays emanating from p partition the plane into four quadrants.

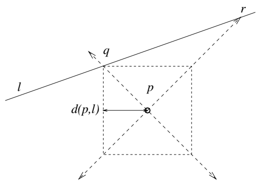


Fig. 2. The L_∞ distance form p to l is $d_x(p,q)$.

points q and r. Assuming that $d_x(p,q) < d_x(p,r)$, the L_∞ distance between p and l is clearly $d_x(p,q)$ (see Fig. 2).

Lemma 1. *If the L_∞ distance from a point p to a line l of slope $\pm s, s \geq 0$ is r, the Euclidean distance from p to l is $S_l r$ where $S_l = \frac{s+1}{\sqrt{s^2+1}}$.*

The *bisector* of two elements (points or lines) is the locus of points equidistant from the two elements. The L_∞ bisector can be regarded as the locus of centers of squares touching the two elements. Figures 3 and 4 illustrate the bisector of two points and two lines respectively. In case of two points along the same horizontal or vertical line the bisector consists of a line segment and two unbounded regions (shaded regions in Fig. 3). Without creating any significant difference we will assign one region to one point and consider only the outermost boundary of the bisecting region as the bisector (thick lines in Fig. 3). The bisectors of two lines have slopes given by the following lemma.

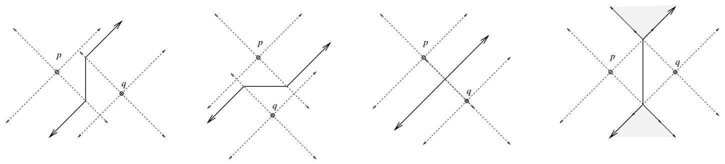


Fig. 3. The L_∞ bisector of two points.

Lemma 2. *The L_∞ bisector of two non-vertical lines l_1 and l_2 of slopes b_1 and b_2, respectively, consists of two lines as follows:*

- *If $b_1 \geq b_2 \geq 0$ the bisector consists of two lines of slopes -1 and $\frac{(b_1+b_2+2b_1b_2)}{(b_1+b_2+2)}$.*
- *If $b_1 \leq b_2 \leq 0$ the slopes of the bisector are $+1$ and $\frac{(-b_1-b_2+2b_1b_2)}{(b_1+b_2-2)}$.*
- *If $b_1 > 0$ and $b_2 < 0$ the bisector has slopes $\frac{b_2-b_1+2b_1b_2}{b_1+b_2}$ and $\frac{b_1+b_2}{b_1-b_2+2}$.*

Note that if one of these two lines is vertical (i.e., $b_1 = \pm\infty$), the L_∞ bisector consists of a 45° line and a line of slope $(1+2b_2)$ for $b_2 > 0$ (resp. $(-1+2b_2)$ for $b_2 < 0$). Moreover if $b_2 = -b_1$ the bisector is a vertical and a horizontal line and if $b_1 = \infty$, $b_2 = 0$ the bisector consists of two 45° lines.

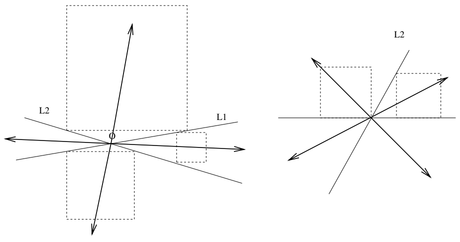


Fig. 4. The L_∞ bisector of two lines.

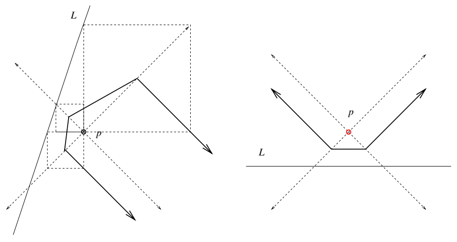


Fig. 5. The L_∞ bisector between a point and a line.

The bisector of a point p and a line l ($p \notin l$, see Fig. 5) consists of at most four parts, one for each quadrant of p. Each part corresponds to the portion of the bisector between l and a vertical or horizontal line through p depending on the quadrant of p. Thus, each part is a line segment or ray whose slopes can be derived by Lemma 2. The unbounded parts of the bisector are always parallel 45° rays (see Fig. 5). For a point p along a non-orthogonal line l the bisector is a 45° line through p and for a point p along an orthogonal line l the bisector is the area enclosed by the 45° lines through p. For simplicity we only consider the boundary of this region as the bisector between p and l.

The L_∞ Voronoi diagram of a set of elements $S = \{e_1, e_2, \ldots, e_n\}$, denoted $\mathcal{V}(S)$, is a partitioning of the plane into regions, called *Voronoi cells*, each of which is associated with an element, called the *owner*, of the cell. The boundary that borders two Voronoi cells is called a *Voronoi edge*, and adjacent Voronoi edges of each Voronoi cell are common to a *Voronoi vertex*. The collection of Voronoi edges and vertices is also called the Voronoi diagram. Fig. 6 illustrates the L_∞ Voronoi diagram in the interior of a simple polygon (the *medial axis*) and Fig. 7 illustrates the L_∞ Voronoi diagram of polygons.

Let G be a *planar straight line graph* on n points in the plane as defined in [1,3] i.e., a set of non-crossing line segments spanned by these points. By the above discussion it is clear that the L_∞ Voronoi diagram of a planar straight-line graph G consists only of straight line edges. Moreover, if the vertices of G are on rational coordinates the L_∞ Voronoi vertices are also rational. Hence, the L_∞ Voronoi diagram is computationally much simpler than its Euclidean counterpart. Most of the existing algorithms to compute the Euclidean Voronoi diagram of segments can be easily modified to compute the L_∞ Voronoi diagram in $O(n \log n)$ time. The well known *incircle test* to determine whether an element is in or out of the circle defined by three other elements (lines or points) now simplifies to a test involving the square defined by the three elements. (Note that three elements need not always define a square). In the worst case the L_∞ *incircle test* corresponds to 1) determining the intersection point I of two lines (bisectors), 2) determining the intersection point J of a line and a 45° line through I, and 3) comparing the horizontal or vertical distances between I and J and between I and one of the defining elements. Note that no square root computations are involved.

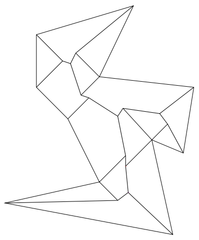


Fig. 6. The L_∞ medial axis of a simple polygon.

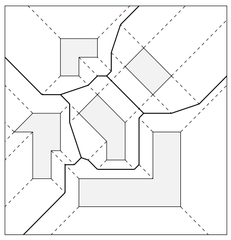


Fig. 7. The L_∞ Voronoi diagram of polygons.

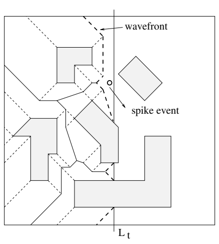


Fig. 8. The L_∞ Voronoi diagram of G_t.

The combinatorial complexity of the L_∞ diagram is similar to the Euclidean one. However, there is some difference in the number of Voronoi vertices produced by reflex angles. The number of 45° rays emanating from a concave vertex v depends on the slopes of the incident edges and can be either one, two, or three. In the Euclidean case the *normals* bisecting a vertex from the incident edges are always two and in the *straight-skeleton* of [1] the corresponding angular bisector is always one. Thus the combinatorial complexity of the L_∞ diagram can be lower or higher than the Euclidean one depending on the input. An exact bound can be derived following the proof of [3] for the Euclidean case.

Lemma 3. *Let G be a planar straight line graph on n points in the plane such that G has t terminals (vertices of degree 1), t_n of which are incident to non-orthogonal edges, r_2 reflex angles inducing two 45° rays, and r_3 reflex angles inducing three 45° rays. The number of (finite and infinite) vertices of the L_∞ Voronoi diagram of G is exactly $2n + t + t_n + r_2 + 2r_3 - 2$.*

Note that in the Euclidean case this bound is $2n + t + r - 2$ where r is the total number of reflex vertices (i.e., $r = r_1 + r_2 + r_3$ for r_1 reflex vertices inducing a single 45° ray) [3]. In the rectilinear case the bound becomes $2n + t - 2$ as for the straight skeletons.

3 A Sweep-Line Algorithm for the L_∞ Voronoi Diagram

In [5], Dehne and Klein described a plane sweep paradigm for the Voronoi diagram of points in "nice metrics". In this section we modify their algorithm to accommodate segments in the L_∞ metric.

Let G be a *planar straight-line graph* on n points. We will compute the L_∞ Voronoi diagram of G, $\mathcal{V}(G)$. Consider a vertical sweep-line L sweeping across the entire plane from left to right. At any instant t of the sweeping process the sweep-line partitions the set of segments (edges) of G into three subsets S_l, S_m and S_r, corresponding to those that lie totally to the left of L, intersect L, and lie totally to the right of L, respectively. The segments in S_m cut L into $|S_m|+1$ sweepline segments, two of which are unbounded, denoted by L_t. Let S_t denote the portions of segments in S_m to the left of L. In other words, $S_t = \{s(t) \mid s \in S_m\}$ where $s(t)$

denotes the portion of s to the left of L. At every instant t of the sweeping process we compute the Voronoi diagram of $G_t = S_l \cup S_t \cup L_t$. Note that at any instant t the sweep-line segments are treated as being part of G. (In case G is a collection of simple polygons and we only want to compute the Voronoi diagram in the exterior of the polygons we exclude from L_t the segments in the interior of polygons, Fig. 8).

The boundary of the Voronoi cell of each segment in L_t is referred to as an *individual wavefront* and the collection of individual wavefronts for all segments in L_t is called the *wavefront*. In Fig. 8, the wavefront is shown in dashed lines. Note that in the case of points the whole wavefront is a single individual wavefront. Clearly, in the L_∞ metric the wavefront is y-monotone. The Voronoi edges that have an endpoint common with the wavefront are called *spike bisectors*.

As the plane sweep proceeds, the wavefront, and therefore the endpoints of spike bisectors, as well as the endpoints of segments in S_t move continuously to the right until an *event* takes place which causes the wavefront to change. Following [5], we have two kinds of events: 1) events corresponding to the occurrence of a vertex or a vertical edge in G, referred to as *site events* and 2) events corresponding to the intersection point of two neighboring spike bisectors, referred to as *spike events*. A site event induced by a point p takes place at $t = x_p$, where x_p is the abscissa of point p. A spike event C takes place at $t = x_c + w$ where x_c is the abscissa of the intersection point and w is the distance of the intersection from the inducing element.

Throughout the plane sweep we implicitly maintain the wavefront in a data structure referred to as the *sweep-line status* which is implemented as a *height-balanced* binary tree $\mathcal{T}$. In particular, the sweep-line status contains the spike bisectors and the edges in S_t. To simulate the wavefront movement to the right we parameterize the endpoints of spike bisectors and segments in S_t as linear functions of t. At any instant t the wavefront is the polygonal line obtained by connecting the endpoints of the elements in $\mathcal{T}$ in increasing y-coordinate value. The events are organized in a priority queue referred to as the *event list* in increasing priority value. Due to space limit we omit the details of the plane sweep algorithm. The number of site and spike events is $O(n)$ and thus the time complexity of the algorithm is $O(n \log n)$.

4 The L_∞ Voronoi Diagram and the Critical Area Problem

The yield of a VLSI chip is the percentage of functional chips among all chips manufactured. Predicting the yield of a chip is important due to the growing need to control the cost of manufacturing. In this section we describe an important problem in VLSI yield prediction that can be solved accurately and efficiently using the L_∞ Voronoi diagram of a layout.

Yield estimation models are based on the concept of critical area which is a measure reflecting the sensitivity of the layout to spot defects caused by particles such as dust and other contaminants in materials and equipment. The main reason for yield loss during manufacturing are "extra material" defects causing shorts between different conducting regions. For information on yield estimation and spot defects see for example [6,7,8,10,11,14,18,19,20].

Given a circuit layout C, the critical area is defined as $A_c = \int_0^\infty A(r)D(r)dr$ where $A(r)$ denotes the area in which the center of a defect of radius r must fall in order to cause circuit failure and $D(r)$ is the density function of the defect size. Defects are usually modeled as circles and the density function has been estimated to follow the "$1/r^3$" distribution [7,14,18,20]. Existing methods require substantial CPU time for large layouts. They can be summarized into three categories: 1) Geometric methods, where $A(r)$ is computed for several different values of r independently and then the results are used to approximate A_c. The methods to compute $A(r)$ are usually based on *shape-expansion* followed by *shape-intersection* (see [20] for references). For rectilinear layouts there is a more efficient scan-line method [14]. 2) Monte Carlo simulation, where a large number of defects is drawn with their radii distributed according to $D(r)$. The critical area is estimated by dividing the number of defects causing faults over the total number of defects [21]. 3) Grid-based approach, where an integer grid is assumed over the layout and the critical radius (i.e., the radius of the smallest defect causing a fault at this point) of every grid point is computed.

For rectilinear layouts our solution to the critical area problem for shorts (assuming that defects are isothetic squares) appears in [13]. It was shown that the problem can be reduced to computing the *2nd order L_∞ Voronoi diagram* of polygons. Consider an arbitrary point p in the layout. The *critical radius* of p for shorts is the radius of the smallest defect which if centered at p overlaps with two different polygons. Such a defect causes a short between the two different conducting regions represented by the polygons. Thus, the critical radius of p is the distance of p from its second nearest polygon. Given the Voronoi diagram of the polygons in the layout, we subdivide the Voronoi cell of a polygon P by the Voronoi diagram of its neighbors. In other words we obtain the *2nd order Voronoi diagram* of polygons. A region of an element s (edge or reflex vertex) within the Voronoi cell of P is defined as $reg_P(s) = \{x \mid d(s,x) \leq d(t,x), \forall t \in C - P\}$, where C is the given collection of polygons. The critical radius of every point $x \in reg_P(s)$ is determined by the distance of x from element s; we say that s is the owner of $reg_P(s)$.

Having the critical area computation done based on Euclidean Voronoi diagrams would be out of the question in practice due to the difficulty of computing the Voronoi diagram of segments. However, the computation in L_∞ is practical. In [13], we showed that once the 2nd order Voronoi diagram is available, the critical area for shorts in rectilinear layouts can be done analytically. In particular, we showed that the critical area formula can be written as a function of the edges of the 2nd order Voronoi diagram. In the following we generalize this result to general non-rectilinear layouts.

4.1 The Critical Area for Shorts as a Function of Voronoi Edges

We have a layer in a circuit layout consisting of a collection of disjoint simple polygons C in arbitrary orientations. Our goal is to evaluate the integral $A_c = \int_0^\infty A(r)D(r)dr$, where $D(r) = r_0^2/r^3$ and r_0 is a minimum optically resolvable size. Recall that $A(r)$ denotes the area of the *critical region* for square defects of radius r. The critical region for radius r is the locus of points where if the center of a square defect of radius r is placed it causes a short i.e., the defect overlaps with two disjoint polygons.

Let's assume that we are given the 2nd order L_∞ Voronoi diagram of C. Fig. 7 shows the L_∞ Voronoi diagram of a set of shapes and Fig. 9 shows the 2nd order subdivision within a bounded cell. Then $A_c = \sum_V A_c(V)$ for all (2nd order) Voronoi cells V where $A_c(V)$ denotes the critical area within V. Note that $A_c(V) = \int_0^\infty A(r,V)D(r)dr$ where $A(r,V)$ denotes the area of the critical region for defect radius r within V.

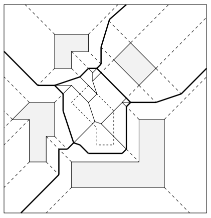


Fig. 9. The 2nd order subdivision within a Voronoi cell.

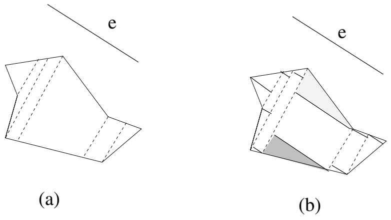


Fig. 10. The decomposition of V into trapezoids.

Let's first concentrate on a single (2nd order) Voronoi cell V having as owner an edge e of slope $\pm s$, $s \geq 0$. The Voronoi cell of a vertex is considered as the Voronoi cell of a vertical or horizontal edge. Consider a decomposition of V into trapezoids by drawing lines perpendicular to e (lines of slope $\pm 1/s$) emanating from the Voronoi vertices of V (see Fig. 10a). Each trapezoid $\mathcal{T}$ is further decomposed into orthogonal triangles and at most one rectangle $\mathcal{R}$ by drawing lines parallel to e (slope $\pm s$) through its vertices. (In case $\mathcal{T}$ is a slanted parallelogram continue the decomposition recursively). We distinguish between two kinds of triangles, *red* and *blue*, depending on the relevant position of the hypotenuse and the orthogonal apex with respect to e. In particular, if the owner e and the orthogonal apex lie on opposite sides of the hypotenuse the triangle is colored red, otherwise it is colored blue. In Fig. 10b, the lightly shaded triangle is red and the darker shaded one is blue.

Given two vertices v_j and v_k such that v_j is closer to e than v_k, let r_j, r_k denote the corresponding critical defect radii i.e., the L_∞ distance of v_j and v_k from the line l through e respectively. We derive the following formulas for the critical area within an element of the above decomposition.

Lemma 4. *The critical area within a rectangle $\mathcal{R}$, a red triangle T_{red} and a blue triangle T_{blue} is given by the following formulas, using the "r_0^2/r^3" defect density distribution:*

$$A_c(\mathcal{R}) = \frac{r_0^2 S}{2}(\frac{l}{r_j} - \frac{l}{r_k}) \tag{1}$$

$$A_c(T_{red}) = \frac{r_0^2 S}{2}(ST \ln(\frac{r_k}{r_j}) - \frac{l}{r_k}) \tag{2}$$

$$A_c(T_{blue}) = \frac{r_0^2 S}{2}(\frac{l}{r_j} - ST \ln(\frac{r_k}{r_j})) \tag{3}$$

where l is the size of the edge of $\mathcal{R}, T_{red}$ and T_{blue} parallel to e, $r_k, r_j, r_k > r_j$ are the maximum and the minimum critical radius of their vertices, $S = \frac{s+1}{\sqrt{s^2+1}}$, where s is the absolute value of the slope of e, and $T = \frac{ts+1}{|t-s|}$, where t is the absolute value of the slope of the hypotenuse (i.e, the slope of the corresponding Voronoi edge). For $s = \infty$, $S = 1$ and $T = t$.

We can derive the critical area within V by adding up the critical areas within every rectangle and triangle in the above decomposition of V. Because of the summation, terms of the form Sl/r_l corresponding to internal decomposition edges cancel out. Similarly, for logarithmic terms involving endpoints of the decomposition other than Voronoi vertices. Thus, the critical area within V can be written as a function of Voronoi edges. Let's color the Voronoi edges that induce the hypotenuse of red triangles as red and those inducing the hypotenuse of blue triangles as blue. Voronoi edges that are incident to a rectangle or induce an orthogonal edge of a triangle must be either parallel or perpendicular to the owner of the cell. Those Voronoi edges that are parallel to the owner are referred to as *prime* and they are portions of bisectors of two parallel edges. Prime Voronoi edges are colored red if the interior and the owner of the cell lie on opposite sides of the edge; otherwise they are colored blue. Those Voronoi edges that are perpendicular to the owner are colored *neutral.* In the formulas of Lemma 4, terms corresponding to red edges get added while terms corresponding to blue edges get subtracted. Neutral edges do not contribute in the formulas. The boundary of the layout is assumed to be a rectangle whose intersection points with the Voronoi diagram are treated as Voronoi vertices. Boundary edges are treated as Voronoi edges but they contribute at most once in the formulas.

We derive the following theorem, which is given without a proof due to space limitation.

Theorem 1. *Given the 2nd order L_∞ Voronoi diagram of polygons of a layer in a circuit layout C and assuming that defects are squares following the "r_0^2/r^3" defect density distribution, the critical area for shorts in that layer is given by the following formula:*

$$A_c = r_0^2(\sum_{\substack{red,\\ prime\ e_i}} \frac{S_i l_i}{r_i} - \sum_{\substack{blue,\\ prime\ e_m}} \frac{S_i l_i}{r_i} + \frac{1}{2}\sum_{\substack{red, nonprime\ e_j,\\ wrt\ e}} S_e^2 T_e \ln\frac{r_k}{r_j} - \frac{1}{2}\sum_{\substack{blue, nonprime\ e_j,\\ wrt\ e}} S_e^2 T_e \ln\frac{r_k}{r_j} + \frac{B}{2})$$

where l_i and r_i denote the length and the critical radius of a prime Voronoi edge e_i, and $r_k, r_j, r_k > r_j$ denote the maximum and the minimum critical radius of a non-prime Voronoi edge e_j. The factors S_e and T_e are as defined in Lemma 4 for each owner e of the non-prime edge e_j. B is the sum of the blue terms for prime boundary edges and appears as a correction factor since boundary edges contribute at most once.

Acknowledgment

I wish to thank Prof. D.T. Lee for valuable contributions.

References

1. O. Aichholzer, F. Aurenhammer, "Straight Skeletons for general Polygonal Figures in the Plane", *Proc. 2nd Int. Computing and Combinatorics Conference,1996* Lecture Notes in Computer Science 1090, 117-126. 9, 10, 12, 13
2. F. Aurenhammer, "Voronoi diagrams: A survey of a fundamental geometric data structure," *ACM Comput. Survey,* 23 1991, 345-405. 9
3. F. Aurenhammer and R. Klein, "Voronoi Diagrams" chapter 18, *Textbook on Computational Geometry*, J.Sack and G. Urrutia (eds), to appear. 9, 12, 13
4. C. Burnikel, K, Melhorn S. Scirra, "How to compute the Voronoi Diagram of Line Segments: Theoretical and Experimental Results" *Proc. 2nd Annu. European Symp. on Algorithms, 1994*, LNCS 855, 227-239. 9
5. F. Dehne and R. Klein, "The Big Sweep": On the power of the Wavefront Approach to Voronoi Diagrams", *Algorithmica*(1997), 17, 19-32. 13, 14
6. A.V. Ferris-Prabhu, "Modeling the Critical Area in Yield Forecast", *IEEE J. of Solid State Circuits*, vol. SC-20, No4, Aug. 1985, 874-878 15
7. A.V. Ferris-Prabhu, "Defect size variations and their effect on the critical area of VLSI devices", *IEEE J. of Solid State Circuits*, vol. SC-20, No4, Aug. 1985, 878-880. 15
8. I. Koren, "The effect of scaling on the yield of VLSI circuits", *Yield Modeling and defect Tolerance in VLSI circuits* W.R. Moore, W. Maly, and A. Strojwas Eds., Bristol UK: Adam-Hilger Ltd., 1988, 91-99 15
9. G. Liotta, F.P. Preparata, and R. Tamassia, " Robust Proximity Queries in Implicit Voronoi diagram", Brown University CS-96-16. 9
10. W. Maly, "Computer Aided Design for VLSI Circuit Manufacturability", *Proc. IEEE*, Feb. 90, 356-392. 15
11. W. Maly, and J. Deszczka, "Yield Estimation Model for VLSI Artwork Evaluation", *Electron Lett.* vol 19, no.6, 226-227, March 1983 15
12. S. N. Meshkat and C. M. Sakkas. " Voronoi diagram for multiply-connected polygonal domains II: Implementation and application", *IBM J. of Research and Development*, Vol. 31, No. 3, May 1987 10
13. E. Papadopoulou, D.T. Lee, "Critical Area Computation – A new Approach", Proc. *International Symposium on Physical Design*, 1998, 89-94. 15
14. J. Pineda de Gyvez, C. Di, "IC Defect Sensitivity for Footprint-Type Spot Defects", *IEEE Trans. on Computer-Aided Design*, vol. 11, no 5, 638-658, May 1992 15
15. V. Srinivasan Personal Communication. 10

16. V. Srinivasan, L.R. Nackman, " Voronoi diagram for multiply-connected polygonal domains II: Algorithm", *IBM Journal of Research and Development*, Vol. 31, No. 3, May 1987 10
17. V. Srinivasan, L.R. Nackman, J.M. Tang, and S.N. Meshkat, "Automatic Mesh Generation Using the Symmetric Axis Transformation of Polygonal Domains", *Proceedings of the IEEE*,Vol. 80, No. 9, Sept. 1992, 1485-1501. 10
18. C.H. Stapper, "Modeling of Defects in integrated circuits photolithographic patterns", *IBM J. Research and Development*, vol.28, no.4, 461-475, 1984. 15
19. C. H. Stapper and R. J. Rosner, "Integrated Circuit Yield Management and Yield Analysis: Development and Implementation" *IEEE Trans. on Semiconductor Manufacturing* Vol. 8, No.2, 1995, 95-101. 15
20. I. A. Wagner and I. Koren, "An Interactive VLSI CAD Tool for Yield Estimation", *IEEE Trans. on Semiconductor Manufacturing* Vol. 8, No.2, 1995, 130-138. 15
21. H. Walker and S.W. Director, " VLASIC: A yield simulator for integrated circuits", *IEEE Trans. on Computer-Aided Design*, vol. CAD-5, no 4, 541-556, Oct. 1986. 15

Facility Location on Terrains[⋆]
(Extended Abstract)

Boris Aronov[1], Marc van Kreveld[2], René van Oostrum[2], and Kasturirangan Varadarajan[3]

[1] Department of Computer and Information Science, Polytechnic University
Brooklyn, NY 11201-3840, USA
aronov@ziggy.poly.edu

[2] Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, Netherlands
{marc,rene}@cs.uu.nl

[3] Department of Computer Science, Duke University
Durham, NC 27708-0129, USA
krv@cs.duke.edu

Abstract. Given a *terrain* defined as a piecewise-linear function with n triangles, and m point *sites* on it, we would like to identify the location on the terrain that minimizes the maximum distance to the sites. The distance is measured as the length of the Euclidean shortest path along the terrain. To simplify the problem somewhat, we extend the terrain to (the surface of) a polyhedron. To compute the optimum placement, we compute the furthest-site Voronoi diagram of the sites on the polyhedron. The diagram has maximum combinatorial complexity $\Theta(mn^2)$, and the algorithm runs in $O(mn^2 \log^2 m(\log m + \log n))$ time.

1 Introduction

Problem statement. A *(polyhedral) terrain* is the graph of a piecewise-linear function defined over a simply-connected subset of the plane. It can be represented by a planar triangulation where each vertex has an associated elevation. The elevation of any point in the interior of an edge (triangle) is obtained by linear interpolation over the two (three) vertices of the edge (resp. triangle). The polyhedral terrain is commonly used as a model for (mountainous) landscapes.

This paper addresses the *facility location problem* for a set of sites on a terrain. More precisely, assume that a set of m point *sites* on a terrain defined by n triangles is given. The distance between two points on the terrain is the minimum length of any path between those points that lies on the terrain. The *facility center* of the sites is the point on the terrain that minimizes the maximum

⋆ B.A. has been partially supported by a Sloan Research Fellowship. M.v.K. and R.v.O. have been partially supported by the ESPRIT IV LTR Project No. 21957 (CGAL). K.V. has been supported by National Science Foundation Grant CCR-93–01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, by matching funds from Xerox Corporation, and by a grant from the U.S.–Israeli Binational Science Foundation. Part of the work was carried out while B.A. was visiting Utrecht University.

distance to a site. We assume throughout that $m \leq n$. To avoid complications involving the boundary of the terrain, we extend the terrain to (the surface of) a polyhedron. We allow only polyhedra homeomorphic to a ball, so that their surfaces are homeomorphic to a sphere. We assume that all faces of the polyhedron have been triangulated. This increases the combinatorial complexity of the polyhedron by at most a constant factor.

Previous results. In the Euclidean plane, the facility center, or the center of the *smallest enclosing disc* of a set of m point sites, can be determined in $O(m)$ time. Several algorithms attain this bound. Megiddo [4] gave the first deterministic linear-time algorithm, and a much simpler, linear expected time algorithm was found by Welzl [10].

In his master's thesis, van Trigt [9] gave an algorithm that solves the facility location problem on a polyhedral terrain in $O(m^4 n^3 \log n)$ time, using $O(n^2(m^2+n))$ space.

There is a close connection between the facility center and the furthest-site Voronoi diagram of the sites. Namely, the facility center must lie at a vertex or on an edge of this diagram. In the plane, with Euclidean distance, the furthest-site Voronoi diagram has cells only for the sites on the convex hull of the set of sites, and all cells are unbounded.

It appears that on a polyhedron, some of the standard properties of furthest-site Voronoi diagrams in the plane no longer hold. For instance, a bisector on the polyhedron is generically a closed curve consisting of as many as $\Theta(n^2)$ straight-line segments and/or hyperbolic arcs, in the worst case. In general, it may also contain two-dimensional portions of the surface of the polyhedron.

Mount [6] showed that the *nearest-neighbor* Voronoi diagram of m sites on (the surface of) a polyhedron with n faces with $m \leq n$ has complexity $\Theta(n^2)$ in the worst case; he also gave an algorithm that computes the diagram in $O(n^2 \log n)$ time. We do not know of any previous work on furthest-site Voronoi diagrams on a polyhedron.

The problem of computing the shortest path between two points along the surface of a polyhedron has received considerable attention; see the papers by Sharir and Schorr [8], Mitchell, Mount and Papadimitriou [5], and Chen and Han [1]. The best known algorithms [1,5] compute the shortest path between two given points, the source s and destination t, in roughly $O(n^2)$ time. In fact, these algorithms compute a data structure that allows us to compute the shortest path distance between the source s to any query point p in $O(\log n)$ time.

New results and methods. This paper gives an $O(mn^2 \log^2 m(\log m + \log n))$ time algorithm to compute the furthest-site Voronoi diagram and find the facility center for a set S of m sites on the surface of a polyhedron with n faces. Given the linear-time algorithm for finding the facility center in the plane, this bound may seem disappointing. However, the algorithm for computing the furthest-site Voronoi diagram is near-optimal, as the maximum combinatorial complexity of the diagram is $\Theta(mn^2)$.

2 Geometric Preliminaries

Previous papers on shortest paths on polyhedra [8,5,1,9] use a number of important concepts that we'll need as well. We review them briefly after giving the relevant definitions.

In the remainder of the paper P is (the surface of) a polyhedron. As stated before, we allow only polyhedra homeomorphic to a ball, so that their surfaces are homeomorphic to a sphere. Let S be a set of m point sites on P. Consider first a single site $s \in P$. For any point p on P we consider a shortest path from p to s; note that in general it is not unique. Such a shortest path has a number of properties. First, if it crosses an edge of P properly, then the principle of reflection holds. This means that if the two incident triangles were pivoted about their common edge to become co-planar, then the shortest path would cross the edge as a straight-line segment. This principle is called *unfolding.* For any vertex on the polyhedron, we define its *total angle* as the sum of the angles at that vertex in each of the triangles incident to it. The shortest path cannot contain any vertex for which the total angle is less than 2π, except possibly at the source p and the target s.

Any shortest path crosses a sequence of triangles, edges, and possibly, vertices. If two shortest paths on the polyhedron cross the same sequence (in the same order), we say that these paths have the same *edge sequence.* If a shortest path from p to s contains a vertex of the polyhedron, the vertex reached first from p is called the *pseudoroot* of p. If the path does not contain any vertex, then site s is called the pseudoroot of p.

The *shortest path map (SPM) of s* is defined as the subdivision of P into connected regions where the shortest path to s is unique and has a fixed edge sequence. For non-degenerate placements of s, the closures of the regions cover P, so the portion of P outside any region, where more than one shortest path to s exists are one-dimensional. It is known that the shortest path map of a site has complexity $O(n^2)$; this bound is tight in the worst case. The SPM restricted to a triangle is actually the Euclidean Voronoi diagram for a set of sites with additive weights. The sites are obtained from the pseudoroots of points on the triangle. The coordinates of the diagram sites are obtained by unfolding the triangles in the edge sequence to the pseudoroot so that they are all co-planar. The weight of a pseudoroot is the distance from the pseudoroot to the site s. It follows that the boundaries of regions in the SPM within a triangle consist of straight-line segments and/or hyperbolic arcs. For any point on a hyperbolic arc or a segment there are two shortest paths to s with different pseudoroots.

Given two sites s and t on the polyhedron, the *bisector* $\beta(s,t)$ is the set of all points on the polyhedron whose shortest path to s has length equal to the shortest path to t. The bisector consists of straight-line segments, hyperbolic arcs, and may even contain two-dimensional regions. Such regions occur only when two sites have exactly the same distance to some vertex of P. For simplicity, we assume that these degeneracies don't occur.

Fix an integer k, $1 \leq k \leq m-1$. For a point $p \in P$, let $S(p,k) \subset S$ be any set of k sites in S for which the shortest path to p is not longer than the path from p to any site in $S \setminus S(p,k)$. The *order-k Voronoi diagram* of a set S of m sites on P is a planar graph embedded in P that subdivides P into open regions such that for any point p in a region, $S(p,k)$ is unique, and such that for any two points p and q in the same region, $S(p,k) = S(q,k)$. The interior of the boundary between two adjacent regions is an *edge* of the order-k Voronoi diagram; it is easy to see that each edge lies on a bisector of two sites in S. The non-empty intersections of the closures of three or more regions of the order-k

Voronoi diagram are its *vertices*. We assume that all vertices have degree three; otherwise, a degeneracy is present.

The order-1 Voronoi diagram is known as the (standard) Voronoi diagram or *closest-site Voronoi diagram*, and the order-$(m-1)$ Voronoi diagram is also called the *furthest-site Voronoi diagram*. In this paper, we only deal with the closest and furthest site Voronoi diagrams, and we give a new algorithm for computing the furthest-site Voronoi diagram of a set S of sites on a polyhedron. We denote it by FVD(S), and refer to it more loosely as *the diagram*. For the closest- (furthest-) site diagrams of S, the *region* $\mathcal{R}(s)$ of a site $s \in S$ is the locus of all points that are closer to (further from) s than to (from) all other points in S.

The following facts are crucial for the algorithm below to work and for the analysis to hold. Lemmas 1, 2, and 3 are similar to the lemmas in Leven and Sharir [3]; they are general statements about a large class of metrics and hold under very general conditions.

Lemma 1. *In the closest-site Voronoi diagram of a set S of sites on P, the region $\mathcal{R}(s)$ of a site $s \in S$ is path-connected.*

Lemma 2. *Bisector $\beta(s,t)$ is connected, and is homeomorphic to a circle.*

Lemma 3. *For any three distinct sites s,t,u, bisectors $\beta(s,t)$, $\beta(s,u)$ intersect at most twice.*

Any family of simple closed curves (in this case, on a topological sphere) every pair of which crosses at most twice is called a *family of pseudocircles*. Thus for every fixed $s \in S$, bisectors $\{\beta(s,t) : t \neq s\}$ form a family of pseudocircles.

Lemma 4. *Bisector $\beta(s,t)$ consists of $O(n^2)$ straight-line segments and hyperbolic arcs.*

Proof. The Voronoi diagram of $\{s,t\}$ can be constructed by techniques similar to those used by Mitchell *et al.* [5] for the computation of the shortest path map of a single site. By adapting their analysis to the case with two sites, one can show that on each triangle of the polyhedron, $\beta(s,t)$ consists of $O(n)$ *elementary arcs* (straight-line segments and hyperbolic arcs). Summing this over all the triangles gives $O(n^2)$. See the paper by Mount [6]. ⊔⊓

Since the edges of the closest- and furthest-site Voronoi diagram lie on the bisectors of two sites in S, each edge also consists of $O(n^2)$ straight-line segments and hyperbolic arcs. To simplify our exposition, the intersections between two adjacent segments or arcs on the edges are referred to as *breakpoints*, as opposed to the *vertices* of the diagram that we defined before. Note that we consider the point where a bisector crosses an edge of P to be a breakpoint.

Lemma 5. *The furthest-site Voronoi diagram FVD(S) of a set S of m sites on a polyhedron has $O(m)$ cells, vertices, and edges.*

Proof. Let $\mathcal{R}_{s>t}$ be the region of points that are further from s than from t, for $s,t \in S$. In this notation $\mathcal{R}(s) = \bigcap_{t\in S, t\neq s} \mathcal{R}_{s>t}$. From Lemma 3 it follows that this intersection is the intersection of a set of pseudo-disks. It follows that for each $s \in S$, the region $\mathcal{R}(s)$ in FVD(S) is connected. So we have at most one cell (region) for each site in S, and, by Euler's relation for planar graphs, the number of vertices and edges of FVD(S) is also $O(m)$. ⊔⊓

We define the *total complexity* of FVD(S) to be the sum of the number of vertices and breakpoints in FVD(S).

Lemma 6. *The maximum total complexity of FVD(S) is $\Theta(mn^2)$.*

Proof. Each edge of FVD(S) is part of some bisector $\beta(s,t)$ for two sites $s,t \in S$. Consequently, the upper bound follows from Lemmas 5 and 4.

As for the lower bound, we describe a construction that shows that FVD(S) for a set S of m point sites on a non-convex polyhedron P with n edges can have complexity $\Omega(mn^2)$. The construction will focus on proving an $\Omega(mn)$-bound for a single edge of P. It is described for point sites in the plane with obstacles. This can then be "lifted" to a non-convex polyhedron.

First we will describe the location of the sites, then the obstacles. Assume that $|S|$ is even; we split S into S_1 and S_2 with $k = m/2$ points each. Figure 1 shows the configuration of the sites $S_1 = \{s_1, \ldots, s_k\}$ (in the figure, $k = 5$). For ease of description, we also specify two additional points s_0 and s_{k+1}; these are *not* sites. The sites $s_1, \ldots, s_k \in S_1$ and the points s_0 and s_{k+1} are placed equally spaced on the lower semi-circle of a circle $\mathcal{C}_1$. For $1 \leq i \leq k+1$, let b_{i-1} be the point where the bisector $\beta(s_{i-1}, s_i)$ meets the upper semi-circle of $\mathcal{C}_1$. Note that any point on the arc of the upper semi-circle $\mathcal{C}_1$ between b_{i-1} and b_i is further away from s_i than from any other site in S_1. Let γ_i denote the cone originating at site s_i that is bounded by the rays ray(s_i, b_{i-1}) and ray(s_i, b_i). The portion of the cone γ_i that lies outside $\mathcal{C}_1$ is further away from s_i than from any other site in S_1. Figure 1 shows just the cones γ_2, γ_3 and γ_4.

Let ℓ be a horizontal line lying some distance above the circle $\mathcal{C}_1$. The second set of sites $S_2 = \{s'_1, \ldots, s'_k\}$ is obtained by reflecting the set S_1 through ℓ. That is, s'_i is the image of s_i from reflecting in ℓ. The points in S_2 lie on a circle $\mathcal{C}'_1$ which is the reflection of $\mathcal{C}_1$. The cone γ'_i is defined analogously and is the reflection of γ_i. Let ℓ_i be the intersection of cone γ_i and ℓ. Note that ℓ_i is also the intersection of γ'_i and ℓ.

We have specified the point sites. Now we will specify the location of the obstacles. The important fact is that the cones $\gamma_i, \ldots, \gamma_k$ have a common intersection around the center of circle $\mathcal{C}_1$. Let $\mathcal{C}_2$ be a small circle lying within this common intersection, and let the segment $\overline{ab}$ be the horizontal diameter of $\mathcal{C}_2$. Figure 1 (detail) shows the circle $\mathcal{C}_2$ and the segment $\overline{ab}$. Let $\overline{a'b'}$ be the reflection of $\overline{ab}$ through ℓ. Our obstacle set will be the segments $\overline{ab}$ and $\overline{a'b'}$ minus a few point holes (through which a path can pass). The segment $\overline{ab}$ has an evenly spaced set $h_1, \ldots, h_n$ of point holes. The segment $\overline{a'b'}$ also has an evenly spaced set $h'_1, \ldots, h'_n$ of point holes; the only difference is that these holes are slightly shifted to the left.

We specified all the points and obstacles. Now, we will argue that the line ℓ is intersected by $k = m/2$ edges of FVD(S), each of which crosses ℓ $\Omega(n)$ times. Let us focus on the portion ℓ_i of the line ℓ. Since any point in ℓ_i is further away from s_i (resp. s'_i) then from any other site in S_1 (resp. S_2), s_i and s'_i are the only interesting sites for ℓ_i. We will now argue that $\beta(s_i, s'_i)$

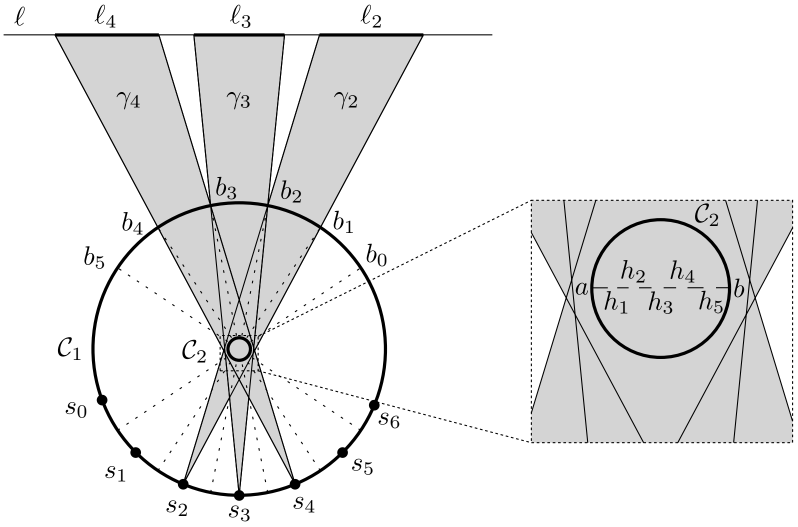


Fig. 1. The configuration of S_1 and the obstacles in $\mathcal{C}_2$ (detail).

crosses ℓ $\Omega(n)$ times. For $1 \leq j \leq n$, let $p_{i,j}$ (resp. $p'_{i,j}$) be the point of intersection of the line through s_i (resp. s'_i) and h_j (resp. h'_j) and the line ℓ. Because of the horizontal shift of the holes in $\overline{a'b'}$, the points occur interleaved on ℓ_i as the sequence $p'_{i,1}, p_{i,1}, p'_{i,2}, p_{i,2}, \dots, p'_{i,n}, p_{i,n}$. This is illustrated in Figure 2 for ℓ_2. For $1 \leq j \leq n$, since s_i can "see" $p_{i,j}$ whereas s'_i cannot, there is a neighborhood around $p_{i,j}$ that is closer to s_i than to s'_i. By symmetric reasoning, there is a neighborhood around $p'_{i,j}$ that is closer to s'_i than to s_i. It follows that the bisector $\beta(s_i, s'_i)$ must cross ℓ_i between $p'_{i,j}$ and $p_{i,j}$, and also between $p_{i,j}$ and $p'_{i,j+1}$. Thus, $\beta(s_i, s'_i)$ crosses ℓ_i $\Omega(n)$ times, as illustrated in Figure 2.

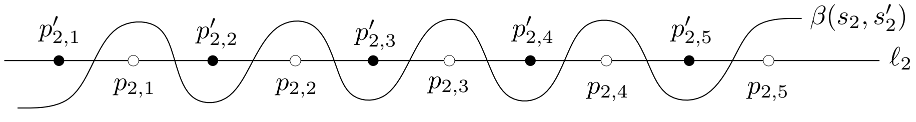


Fig. 2. Detail of $\beta(s_2, s'_2)$.

One gets $\Omega(kn) = \Omega(mn)$ crossings for line ℓ, $\Omega(n)$ for each ℓ_i. The pattern can be repeated on n lines parallel to ℓ and sufficiently close to ℓ. This gives $\Omega(mn)$ crossings for each of the n lines. The sites and the obstacles can be perturbed to a general position without affecting the lower bound complexity. By treating the lines as edges on a polyhedron, and 'raising vertical cylinders' with the obstacles as bases, we can get the $\Omega(mn^2)$ bound for a polyhedron. ⊔⊓

Using standard arguments, and the fact that FVD(S) has maximum total complexity $O(mn^2)$, we obtain the following.

Corollary 1. *Given FVD(S), the facility center of S can be computed in $O(mn^2)$ time.*

3 Computing the Furthest-Site Voronoi Diagram

In this section, we describe our algorithm for computing the furthest-site Voronoi diagram of the given set S of sites on the polyhedron P. Our algorithm uses ideas from the algorithm of Ramos [7] for computing the intersection of unit spheres in three dimensions. We first give an outline of the algorithm, and get into the details in the subsequent subsections.

The algorithm for computing FVD(S) works as follows:

- As a preprocessing step, compute the shortest path map for every site in S.
- Subdivide S into two subsets R (the "red" sites) and B (the "blue" sites) of about equal size.
- Recursively compute FVD(R) and FVD(B).
- Merge FVD(R) and FVD(B) into FVD($R \cup B$) = FVD(S) as follows:
  - Determine the set of sites $R_0 \subset R$ that have a non-empty region in FVD(R), i.e., FVD(R) = FVD(R_0). Observe that the remaining sites in $R \setminus R_0$ don't influence the final diagram. Similarly, compute $B_0 \subset B$.
  - Determine an *low-degree independent set* $M \subset R_0$, which is a subset with the property that the region of a site $s \in M$ has at most 10 neighbors in FVD(R_0), and no two sites $s, s' \in M$ are neighbors in FVD(R_0). (Two sites are said to be *neighbours* if their regions share an edge of the diagram.) Compute $R_1 = R_0 \setminus M$ and FVD(R_1), and repeat this step to generate a Dobkin-Kirkpatrick hierarchy [2] $R_0 \supset R_1 \supset \ldots \supset R_k$ and their furthest-site Voronoi diagrams, such that R_k has only a constant number of sites. Do the same for the blue sites to achieve $B_0 \supset B_1 \supset \ldots \supset B_l$ and their furthest-site Voronoi diagrams.
  - Compute FVD($R_i \cup B_l$) for $0 \leq i \leq k$, exploiting the fact that B_l has only a constant number of sites. Similarly, compute FVD($R_k \cup B_j$) for $0 \leq j \leq l$. This is the *basic merge step.*
  - Compute FVD($R_i \cup B_j$) from FVD($R_i \cup B_{j+1}$) and FVD($R_{i+1} \cup B_j$). This is the *generic merge step*, which when repeated gives FVD($R_0 \cup B_0$) = FVD(S).

3.1 Constructing the Hierarchy for R_0 and B_0.

We describe how to compute the hierarchy $R_0 \supset R_1 \supset \ldots \supset R_k$ and their furthest-site Voronoi diagrams; for B_0, this is done analogously. The computation is similar to the Dobkin-Kirkpatrick hierarchy [2]. Using the fact that furthest-site Voronoi diagrams and their dual graphs are planar graphs, we can show that there is a low-degree independent set $M \subset R_0$ such that M contains a constant fraction of the sites of R_0. In fact, such an independent set can be found in $O(|R_0|)$ time using a greedy algorithm [2].

As for the computation of $\text{FVD}(R_1) = \text{FVD}(R_0 \setminus M)$, we remove the sites in M one at a time from R_0 and update the diagram after the removal of each site. Let s be such a site in M, and let p be a point that lies in the region in $\text{FVD}(R_0)$ of s. After updating the diagram, p must lie in the region of a site s' that is a neighbour of s in $\text{FVD}(R_0)$. So the region of s is divided among its neighbors, of which there are only a constant number, and all diagram edges in that region lie on the bisectors of those neighbors. Computing the bisector for two sites takes $O(n^2 \log n)$ using the techniques from Mitchell, Mount and Papadimitriou[5], and computing the bisector for every pair of neighbors of s takes asymptotically the same time. Given these bisectors, we can easily trace the edges of the diagram in the region of s in $O(n^2 \log n)$ time, using the vertices of that region as starting points.

After all the sites in M have been removed from R_0 and $\text{FVD}(R_1)$ has been constructed, we repeat this procedure to create $\text{FVD}(R_2), \ldots, \text{FVD}(R_k)$. The total number of diagrams we construct this way is $O(\log m)$.

By lemma 6, the size of $\text{FVD}(R_i) = O(|R_i|n^2)$.

Since $\sum_{i=0}^{k} |R_i|$ is a geometric series, the total time for computing all independent sets is $O(m)$. The computation of the bisectors of neighbors and the reconstruction of the diagram after the removal of a single site from a diagram takes $O(n^2 \log n)$ time, and the total number of sites removed is less than m. It follows that the construction of the the hierarchy $R_0 \supset R_1 \supset \ldots \supset R_k$ and their furthest-site Voronoi diagrams takes $O(mn^2 \log n)$ time in total. The total size of all diagrams constructed is $O(mn^2)$.

3.2 The Generic Merge Step

The generic merge step is the computation of $\text{FVD}(R_i \cup B_j)$ from $\text{FVD}(R_i \cup B_{j+1})$ and $\text{FVD}(R_{i+1} \cup B_j)$, which when repeated gives the required $\text{FVD}(R_0 \cup B_0) = \text{FVD}(S)$. First some terminology: we call the sites in R_{i+1} the *old* red sites, and the sites in $R_i \setminus R_{i+1}$ the *new* red sites. Similarly, the sites in B_{j+1} are the old blue sites, and the sites in $B_j \setminus B_{j+1}$ are the new blue sites. Now consider any vertex v of $\text{FVD}(R_i \cup B_j)$. The important fact is that not all three Voronoi regions incident to that vertex correspond to new sites; there must be at least one old red or blue site whose face is incident to v, because new red (blue) regions form an independent set in $\text{FVD}(R_i)$ (resp. $\text{FVD}(B_j)$). So to determine all the vertices of $\text{FVD}(R_i \cup B_j)$, it suffices to compute the regions in $\text{FVD}(R_i \cup B_j)$ of all old red and blue sites.

Consider an old red site r. The region of r in $\text{FVD}(R_i \cup B_{j+1})$ contains all points that are further from r than from any other site in $R_i \cup B_{j+1}$, and the region of r in $\text{FVD}(R_{i+1} \cup B_j)$ contains all points that are further from r than from any other site in $R_{i+1} \cup B_j$. The region of r in $\text{FVD}(R_i \cup B_j)$ is therefore the intersection of its regions in $\text{FVD}(R_i \cup B_{j+1})$ and $\text{FVD}(R_{i+1} \cup B_j)$. We can compute this intersection for each face of the polyhedron separately by a line-sweep of the regions of r in $\text{FVD}(R_i \cup B_{j+1})$ and $\text{FVD}(R_{i+1} \cup B_j)$. The time needed for computing the vertices of $\text{FVD}(R_i \cup B_j)$ is therefore bounded by $O(C \log C)$, where $C = \max(n^2|R_i \cup B_{j+1}|, n^2|R_{i+1} \cup B_j|, n^2|R_i \cup B_j|)$, which in turn equals $n^2(|R_i| + |B_j|)$. Hence, computing the vertices of $\text{FVD}(R_i \cup B_j)$ takes $O(n^2(|R_i| + |B_j|) \log(n^2(|R_i| + |B_j|))) = O(n^2(\log n + \log m)(|R_i| + |B_j|))$.

The edges of $\mathrm{FVD}(R_i \cup B_j)$ are either edges incident to the faces of old red or blue sites (which we already computed), or edges between the faces of two new sites of the same color (these edges are sub-edges of edges in $\mathrm{FVD}(R_i)$ or $\mathrm{FVD}(B_j)$, and can easily be found), or they are edges between the faces of a new red and a new blue site. For the latter category of edges we already have the incident vertices computed, and we can trace the edges after computing the bisector of the new red and new blue site. The total number of bisectors we have to compute and trace is bounded by $|R_i \cup B_j|$, so this takes $O(n^2 \log n(|R_i|+|B_j|))$ time. We conclude that computing $\mathrm{FVD}(R_i \cup B_j)$ from $\mathrm{FVD}(R_i \cup B_{j+1})$ and $\mathrm{FVD}(R_{i+1} \cup B_j)$ takes $O(n^2(\log n + \log m)(|R_i| + |B_j|))$ time.

Summing this over all $0 \leq i \leq k$, $0 \leq j \leq l$ gives

$$O(n^2(\log n + \log m)\sum_{i=0}^{k}\sum_{j=0}^{l}(|R_i| + |B_j|)) \tag{1}$$

We have

$$\sum_{i=0}^{k}\sum_{j=0}^{l}|B_j| = O(\sum_{i=0}^{k}|B_0|) = O(k|B_0|) = O(m \log m), \tag{2}$$

and similarly $\sum_{i=0}^{k}\sum_{j=0}^{l}(|R_i|)$ is $O(m \log m)$. It follows that the total time spent in all the iterations of the generic merge step is $O(mn^2 \log m(\log m + \log n))$.

3.3 The Basic Merge Step

In the basic merge step, we compute $\mathrm{FVD}(R_i \cup B_l)$ for $0 \leq i \leq k$, and $\mathrm{FVD}(R_k \cup B_j)$ for $0 \leq j \leq l$. We will exploit the fact that B_l and R_k contain only a constant number of sites. We will only describe the computation of $\mathrm{FVD}(R_i \cup B_l)$ for a fixed i; all other diagrams are computed similarly.

1. For each site $r \in R_i$ and $b \in B_l$, we compute the region of r in $\mathrm{FVD}(\{r, b\})$. To do this, we compute the *closest-site* Voronoi diagram for sites r and b using the $O(n^2 \log n)$ algorithm of Mount [6]. The region of r in $\mathrm{FVD}(\{r, b\})$ is clearly the region of b in the closest-site diagram. The total time for all pairs r and b is $O(|R_i|n^2 \log n)$, since there are only $O(|R_i|)$ pairs.
2. Next, we compute the region of each site $r \in R_i$ in $\mathrm{FVD}(r \cup B_l)$; to do this, we intersect the regions of r in $\mathrm{FVD}(\{r, b\})$ over all $b \in B_l$. Since B_l has only a constant number of sites, the intersection can be accomplished in $O(n^2 \log n)$ time for a single red site $r \in R_i$ by iterating the intersection procedure in the generic merge step. The time taken for all the sites in R_i is $O(|R_i|n^2 \log n)$.
3. Next, we compute the region of each site $r \in R_i$ in $\mathrm{FVD}(R_i \cup B_l)$ by intersecting its regions in $\mathrm{FVD}(r \cup B_l)$ and $\mathrm{FVD}(R_i)$. The intersection procedure is similar to the one in the generic merge step, and it can be shown that its running time for all sites in R_i is $O(|R_i|n^2 \log n)$.
4. To complete the computation of $\mathrm{FVD}(R_i \cup B_l)$, it remains to compute the regions of the blue sites. Note that all the vertices of $\mathrm{FVD}(R_i \cup B_l)$ are known at this stage; these are either vertices on the boundary of the red regions, or

they are the vertices of $\mathrm{FVD}(B_l)$ that have 'survived' after the computation of the red sites. The edges of $\mathrm{FVD}(R_i \cup B_l)$ are now traced out just as in the generic merge step; in fact, the situation is much simpler here. The time taken in this step is $O(|R_i|n^2 \log n)$.

Putting everything together, the time complexity of computing $\mathrm{FVD}(R_i \cup B_l)$ is $O(|R_i|n^2 \log n)$. Hence, the time needed for computing all the diagrams in the basic merge step is $O(mn^2 \log n)$.

3.4 Total Running Time and Memory Requirements

The time for merging $\mathrm{FVD}(R)$ and $\mathrm{FVD}(B)$ into $\mathrm{FVD}(R \cup B)$ is dominated by the generic merge step, which requires $O(mn^2 \log m(\log m + \log n))$ time; the total running time satisfies the recurrence

$$T(m) = T(\lfloor m/2 \rfloor) + T(\lceil m/2 \rceil) + O(mn^2 \log m(\log m + \log n)) \tag{3}$$

which solves to $T(m) = O(mn^2 \log^2 m(\log m + \log n))$.

The memory requirements of the algorithm are linear in the size of all diagrams that are constructed in the process, which is $O(mn^2 \log m)$.

4 Discussion and Conclusions

We have shown that the furthest-site Voronoi diagram of a set S of m sites on the surface of a polyhedron P has complexity $\Theta(mn^2)$, and we have given an algorithm for computing the diagram in $O(mn^2 \log^2 m(\log m + \log n))$ time. Once the diagram has been computed, the facility center, which is the point on P that minimizes the maximum distance to a site in S, can be found in $O(mn^2)$ time by traversing the edges of the diagram.

The merge step in our divide-and-conquer approach for the computation of $\mathrm{FVD}(S)$ is quite complicated, and it would be pleasant to find a simpler method. Merging the recursively computed diagrams by sweeping seems natural, but the number of intersections of edges of both diagrams can be superlinear (in m), while only a linear number of them can end up as a vertex of the resulting diagram.

It would be a challenge to find an output-sensitive algorithm, i.e., an algorithm that takes time proportional to the number edges/vertices in the diagram plus the number of their intersections with the edges of P. Even more ambitious would be the computation of the diagram without explicitly representing all intersections of Voronoi edges and edges of the polyhedron.

Another interesting issue is approximation: find (in $o(mn^2)$ time) a point with the property that the distance to the furthest site is at most $(1 + \varepsilon)$ times the radius of the smallest enclosing circle.

Finally, it is worth investigating if the facility location problem can be solved without constructing the furthest-site Voronoi diagram. Recall that the facility location problem in the plane can be solved using techniques related to fixed-dimensional linear programming [4,10].

References

1. J. Chen and Y. Han. Shortest paths on a polyhedron. *Internat. J. Comput. Geom. Appl.*, 6:127–144, 1996. 20
2. D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6:381–392, 1985. 25
3. D. Leven and Micha Sharir. Intersection and proximity problems and Voronoi diagrams. In J. T. Schwartz and C.-K. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 187–228. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987. 22
4. N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983. 20, 28
5. Joseph S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987. 20, 22, 26
6. D. M. Mount. Voronoi diagrams on the surface of a polyhedron. Technical Report 1496, Department of Computer Science, University of Maryland, 1985. 20, 22, 27
7. E. Ramos. Intersection of unit-balls and diameter of a point set in R^3. *Computat. Geom. Theory Appl.*, 6:in press, 1996. 25
8. Micha Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM J. Comput.*, 15:193–215, 1986. 20
9. M.J. van Trigt. Proximity problems on polyhedral terrains. MSc. thesis, Dept. Comput. Sci., Utrecht University, 1995. INF/SCR-95-18. 20
10. Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes Comput. Sci.*, pages 359–370. Springer-Verlag, 1991. 20, 28

Computing Weighted Rectilinear Median and Center Set in the Presence of Obstacles*

(Extended Abstract)

Joonsoo Choi[1], Chan-Su Shin[2], and Sung Kwon Kim[3]

[1] Dept. of Comp. Sci., Kookmin U., KOREA
jschoi@kmu.kookmin.ac.kr
[2] Dept. of Comp. Sci., KAIST, KOREA
cssin@jupiter.kaist.ac.kr
[3] Dept. of Comp. Sci. Eng., Chung-Ang U., KOREA
ksk@point.cse.cau.ac.kr

1 Introduction

Given a set $\mathcal{B}$ of obstacles and a set S of source points in the plane, the problem of finding a set of points subject to a certain objective function with respect to $\mathcal{B}$ and S is a basic problem in applications such as facility location problem [5].

Suppose that each source point $s \in S$ has a positive weights $w(s)$. The *median problem* is to find a point minimizing the following objective function

$$D(t) = \sum_{s \in S} w(s)d(s,t), \tag{1}$$

and the *center problem* is to find a point minimizing

$$E(t) = \max_{s \in S} w(s)d(s,t), \tag{2}$$

where $d(s,t)$ is the length of an obstacle-avoiding shortest path between two points s and t. A point t minimizing objective function (1), (2) is called a *median*, *center* point of S with respect to $\mathcal{B}$, respectively. The set of all median points is called the *median set* of S and the set of all center points the *center set* of S. See Figure 1. In the following, we assume that the distance metric is rectilinear.

In the absence of obstacles, optimal algorithms for many variants of median and center problems have been known. In 1-dimensional case, it is well known that median set is a point coinciding with a source point or all points between two consecutive source points [5] and the median set can be computed in linear time. In the plane, the median set can be computed by reducing the problem to a pair of 1-dimensional problems [5]. The center problem in the plane can also be solved by reducing it to a pair of 1-dimensional problems which can be solved in linear time [5,3].

In the presence of obstacles, there have been only few algorithms for median and center problems. When all source points are in a simple rectilinear

* This work is partially supported by KOSEF, Grant No. 98-0102-07-01-3.

polygon P, Chepoi and Dragan [2] showed that a median point can be computed in $O(n + m \log n)$ time where m is the number of source points and n is the number of vertices of P. Given a set of polygonal obstacles, Larson and Sadiq [8] gave an algorithm computing a median point. Recently, Kusakari and Nishizeki [7] presented an output-sensitive algorithm that computes weighted median set in the presence of pairwise-disjoint, axis-parallel rectangular obstacles. For a small number of source points, the time complexity of their algorithm is $O((n + k) \log n)$, where n is the number of obstacles and k is the complexity of the median set. However, no algorithms for the center problem have been known in the presence of pairwise-disjoint, axis-parallel rectangular obstacles. In addition to the problems we cited, many variants of the rectilinear median and center problems have been studied [6,10].

In this paper, we present algorithms computing the weighted median set and center set of source points in the presence of pairwise-disjoint, axis-parallel rectangular obstacles. For a small number of source points, the time complexity of our algorithm is $O(n \log n + k)$ for median set problem, where n is the number of obstacles and k is the complexity of the median set. For center set problem, we investigate the worst-case size of a center set and present an algorithm computing the center set in $O(n^2)$ time when the number of source points is fixed.

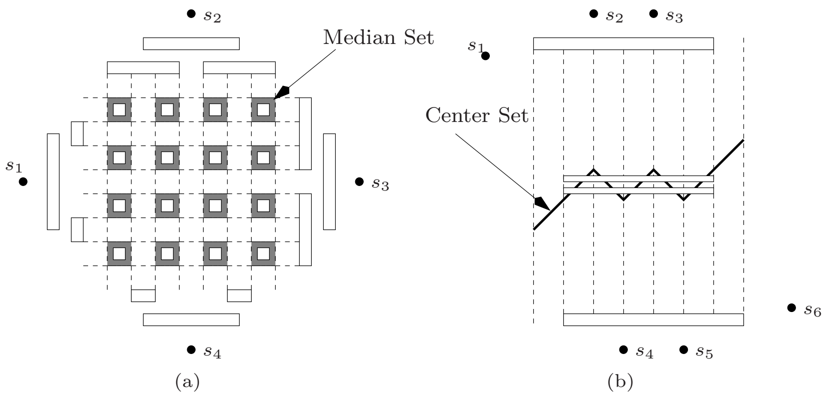


Fig. 1. (a) Median set of 4 source points with respect to 28 rectangular obstacles. (b) Center set of 6 source points with respect to 4 rectangular obstacles.

2 Preliminaries

Let X, Y be the coordinate axes of $\mathbb{R}^2$. For a point $p \in \mathbb{R}^2$, let p_x, p_y denote the X- and Y-coordinates of p. The L_1 (or *rectilinear*) distance between two points p and q is $\|p - q\|_1 := |p_x - q_x| + |p_y - q_y|$.

We consider only rectilinear paths in the plane, simply calling them "paths". Such a path is specified by a sequence of points $\pi = (p_0, p_1, \ldots, p_k)$, where p_i and p_{i+1} differ in exactly in one coordinate. The path π has a length $\ell(\pi) = \sum_{i=1}^{k} \|p_i - p_{i-1}\|_1$. If ϕ is a coordinate axis, we say a path is *ϕ-monotone* if the ϕ-coordinate of points along the path is monotone non-decreasing or non-increasing. A *monotone path* is one that is X-monotone and Y-monotone.

The closure of the set, $\mathbb{R}^2$ minus the union of the obstacles in $\mathcal{B}$, is called the *free space* FP($\mathcal{B}$) of $\mathcal{B}$. Points or paths in FP($\mathcal{B}$) are said to be *free*. For two free points u, v, a shortest path $\pi(u, v)$ between them is a free path that has minimum path length among all possible free paths connecting u and v. The length of $\pi(u, v)$ is denoted by $d(u, v)$ and is called the *distance* between u and v. Considering $\pi(u, v)$, the sum of lengths of its segments parallel to the X-coordinate axis is called the *x-distance* between u and v, and is denoted by $d_x(u, v)$. Similarly, the y-distance, $d_y(u, v)$, between u and v is defined.

Pyramids For a coordinate direction $\phi \in \{\pm X, \pm Y\}$, we call a path a ϕ-path if is a monotone path $(p_0, p_1, \ldots, p_k)$ (possibly $p_k = \infty$) where p_0 is a source point $s \in S$ and the path segment $[p_0, p_1]$ is ϕ-directed and maximal in the sense that p_1 is either touches some obstacle or goes to ∞ (then $k = 1$). If it touches some rectangle B, then $[p_1, p_2]$ is orthogonal to the coordinate axis $|\phi|$ and p_2 is the corner of B. The next segment is again directed as ϕ and is maximal. This pattern is repeated alternatively. It is easily seen that $[p_{2i-1}, p_{2i}]$ (for all $i \geq 1$) must consistently be directed along some direction ϕ'. If $\phi = +X$ then ϕ' is either $+Y$ or $-Y$. We may call the $(+X)$-path an *East-North staircase* if $\phi' = +Y$, and an *East-South staircase* if $\phi' = -Y$. For each source point $s \in S$, the $(+X)$-pyramid of $\mathcal{B}$ relative to s is the rectilinear convex region bounded by the East-North and the East-South staircases. We similarly define the $(-X)$-pyramid, the $(\pm Y)$-pyramids. The subset of $\mathbb{R}^2$ outside of the four pyramids are called the *monotone region* relative to s.

The basis for our algorithms is the following simple but crucial observation. The proof can be derived from the results by De. Rezende et al. [4].

Lemma 1. *For any two points p, q on a line-segment $L \subset FP(\mathcal{B})$, and a source point s,* (i) *$d_y(s, p) = d_y(s, q)$, if L is horizontal;* (ii) *$d_x(s, p) = d_x(s, q)$, if L is vertical.*

Valleys and Ridges For a free point $p = (x, y)$, let $f_{s,\mathcal{B}}(x, y) = f_{s,\mathcal{B}}(p) = d(s, p)$ (or simply $f_s(x, y), f_s(p)$) be a distance function between a source point s and p. For a horizontal line-segment L in free space, and a point $p = (x, b) \in L$, let $f_s^L(x) = f_s(x, b)$. For a vertical free line-segment L, a function $f_s^L(y)$ is defined similarly.

A free point p on a horizontal line-segment L is called a *valley point* (*ridge point*) of s if $f_s^L(x)$ has a local minimum (maximum) at $x = p_x$, respectively. Similarly we define a valley, ridge point on a vertical line-segment. Note that there is no valley, ridge point in the monotone region relative to s. A set of valley (ridge) points of s makes a line-segment which is called a *valley* (*ridge*) of s, respectively. See Figure 2.

Now we briefly explain how to compute ridges and valleys in the $(+Y)$-pyramid relative to s. We sweep a horizontal line L in $(+Y)$-direction from s stopping when L overlaps horizontal edges of obstacles in the $(+Y)$-pyramid. We keep a set of valley and ridge points on L. A binary search tree is used to maintain a sequence of the points from left to right on L. The set of data is updated at each stop position of L which is each edge of obstacles. The trajectory of each valley (ridge) point on L while L moves in $(+Y)$-direction creates a vertical valley (ridge), respectively. As moving L in the $+Y$ direction starting from s, we can compute all valleys and ridges in $O(n \log n)$ time. We will omit the details in this version.

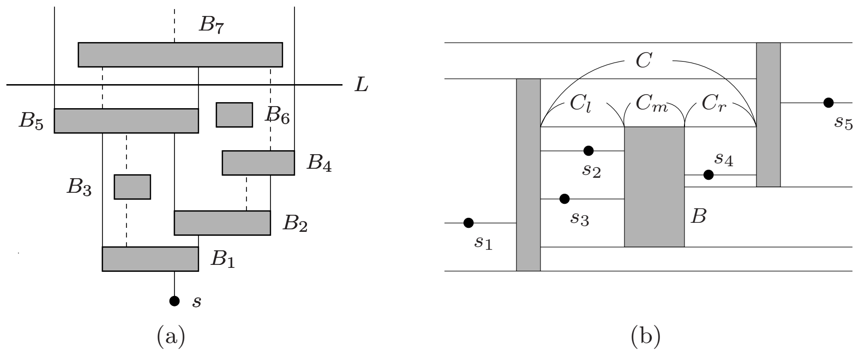


Fig. 2. (a) Valleys (solid lines) and ridges (dotted lines) within the $(+Y)$-pyramid relative to s. (b) Ridges relative to s_1, s_5 overlap a cut C of B.

Define a *cut* of a rectangle B as an extension of an edge of B through free space in both directions until it first encounters other rectangles. A cut may extend to infinity. Similarly we define horizontal and vertical cuts of a source point. Assuming the general position of rectangles and source points, each cut of a rectangle B is partitioned into three parts, called *sub-cuts*, by B: Suppose that C is a horizontal cut of B. One sub-cut is the same as an edge of B and it is called the *middle sub-cut* of C. The other two sub-cuts lie outside of B and they are called the *left sub-cut*, *right sub-cut* of C, respectively. In Figure 2 (b), C_m is the middle sub-cut of a horizontal cut C. Left, right sub-cuts of C are represented by C_l, C_r, respectively.

Considering set of ridges and valleys of all source points in S, each valley of a source point is always represented by a union of sub-cuts and some ridges may overlap cuts. In Figure 2 (b), a cut C is a ridge of s_1 and $C_l \cup C_m$ is a valley of s_4. Furthermore ridges and valleys of source points may partially overlap a sub-cut even we assume the general position of obstacles and source points. In Figure 2 (b), sub-cut C_m partially overlap valleys of s_2, s_3, s_4 and ridges of s_1, s_5.

For a sub-cut C, let $S_v(C), S_r(C)$ be a set of source points that generate valleys, ridges partially overlapping C, respectively. C is called a *valley, ridge, null* sub-cut of S if $W(C) = \sum_{s\in S_v(C)} w(s) - \sum_{s\in S_r(C)} w(s)$ is positive, negative, zero, respectively.

3 Computing the Median Set in the Plane

3.1 The Location of Median Set

By valleys of all source points in S and edges of obstacles, $\mathrm{FP}(\mathcal{B})$ is partitioned into subregions. We call the partition the *valley-partition*, $\mathrm{VP}(S)$, and call each subregion a *valley-subregion*. For a valley-subregion of $\mathrm{VP}(S)$, there may be holes caused by a presence of obstacles which are called *void obstacles*. It is easy to see that the absence of void obstacles does not cause any change to the distance between a free point and any source point in S. Therefore, in this subsection, we assume that there is no void obstacle. Thus each valley-subregion of $\mathrm{VP}(S)$ is a simple rectilinear polygon.

For a horizontal line-segment L in free space, points that have the minimum sum of distances to all source points among points on L are called *candidate median points* on L. Let $L = [p, q]$ be a maximal horizontal line-segment in free space where end-points p and q may lie on the boundary of obstacles or lie at ∞. For L, let $p_1, p_2, \ldots, p_m$ be intersections of L with vertical valleys or ridges of source points except that p_1, p_m are end-points of L. Suppose that the points are sorted in ascending order of their x-coordinates. Consider three consecutive points p_{i-1}, p_i, p_{i+1} on L. Let $I = [p_{i-1}, p_{i+1}]$. Then S is partitioned into four subsets S_1, S_2, S_3, S_4 such that each source point $s \in S_1$ has a distance function $f_s^L(p)$ from s to $p \in I$ that is monotone decreasing in I. Each source point in S_2 has a monotone increasing distance function. Source points in S_3, S_4 create valley, ridge points at p_i, respectively.

For each interval $I_i = [p_i, p_{i+1}]$, called a *base-interval* on L, the distance function $f_s^{I_i}(p)$ between any source point $s \in S$ to $p \in I_i$ is a linear function. Thus candidate median points of I_i are p_i, p_{i+1} or all points in I_i.

Suppose that p_i is an intersection of L with only ridges of source points in S. Then sum of distance functions in the interval I is strictly concave and p_i cannot be a candidate median point of I. Therefore on a horizontal line-segment L, candidate median points lie on intersections of vertical valleys or all points in base-intervals whose end-points are intersections of vertical valleys of source points in S.

Suppose that all points in two consecutive base-intervals I_{i-1}, I_i are candidate median points on L. Then the sum of distances from a point $p \in I$ to all source points in S is

$$D(p) = \sum_{s\in S_1} f_s^L(x) + \sum_{s\in S_2} f_s^L(x) + \sum_{s\in S_3} f_s^L(x) + \sum_{s\in S_4} f_s^L(x)$$

$$= \sum_{s\in S_1} w(s)(d(p_{i-1},s) - (x-(p_{i-1})_x)) + \\ \sum_{s\in S_2} w(s)(d(p_{i-1},s) + (x-(p_{i-1})_x)) + \\ \sum_{s\in S_3} w(s)(d(p_i,s) + |x-(p_i)_x|) + \sum_{s\in S_4} w(s)(d(p_i,s) - |x-(p_i)_x|). \quad (3)$$

The first two terms in the right part of Equation 3 are linear in the domain, and the third term is strictly convex and the fourth term is strictly concave. Since all points in I are median points, we have the condition $D(p_{i-1}) = D(p_i) = D(p_{i+1})$. This condition will be satisfied if $\sum_{s\in S_3} w(s) = \sum_{s\in S_4} w(s)$, i.e., the sub-cut intersecting L at p_i is a null sub-cut. Thus we have:

Theorem 1. (i) *The median set of source points S consists of* (1) *vertices of valley-subregions of VP(S) which are intersections of vertical and horizontal valleys of S;* (2) *edges of valley-subregions of VP(S) which are subset of valleys of S;* (3) *all points in some valley-subregions of VP(S).* (ii) *Suppose that all points in two adjacent valley-subregions of VP(S) separated by a sub-cut C are median points. Then C is a null sub-cut.*

From Theorem 1, at least one intersection of vertical and horizontal valleys of source points in S is a median point. Every valley of source points lies on a cut of source points and obstacles. Furthermore the median set of S is a union of vertices, edges of valley-subregions of VP(S), or all points in some valley-subregions of VP(S). Thus the complexity of VP(S) is at most the complexity of the partition of free space by cuts of obstacles and source points. Since the number of cuts is $O(n+m)$, the upper bound of the worst-case complexity of the median set is $O((n+m)^2)$. For the lower bound, we can construct an example whose median size is $\max(n^2, nm, m^2) = \Omega((n+m)^2)$; the construction is appeared in the full version. Therefore we have:

Corollary 1. (i) *There exists at least one median point among intersections of cuts of source points and obstacles.* (ii) *The worst-case complexity of the median set is $\Theta((n+m)^2)$.*

3.2 Computing One Median Point

Before computing the median set of source points S, we first compute the optimum value D^* of the objective function (1) in Introduction. By Corollary 1, there exists at least one median point among the intersections of cuts of source points and obstacles. For each horizontal cut L, we compute an intersection point p, among intersections with vertical cuts, that has a minimum sum of distances to all source points. By Lemma 1, points on L have the same y-distances to all source points. Therefore the point p is the intersection of L with the vertical cut having the minimum x-distances to all source points among the vertical cuts intersecting L.

For every horizontal cut, we do the following to compute efficiently a vertical cut having the minimum sum of x-distances. We obtain a decomposition $P(\mathcal{B}, S)$ of free space into rectangular subregions using horizontal cuts of obstacles and source points. Each subregion of P is called a *horizontal strip*. With the partition P, there is a dual graph $G(P)$, called the *horizontal partition graph* (HPG) of P: Vertices of $G(P)$ are the horizontal strips of P and two vertices are adjacent in $G(P)$ iff the corresponding strips in P are bounded by a common sub-cut. Assuming the general position of obstacles and source points, the maximum degree of vertices in $G(P)$ is four.

Edges of $G(P)$ are partitioned into three classes: Edges corresponding to cuts of source points are called *crossing edges*. Pairs of edges corresponding to left, right sub-cuts of cuts extending bottom (top) edges of obstacles are called *merging* (*splitting*) edges, respectively.

Scheme of the Algorithm Initially we compute the set of horizontal, vertical cuts of obstacles and source points. Next for each horizontal (vertical) cut C, we compute the sum, $D_y(C, S)$ (($D_x(C, S)$), of y-distances (x-distances) to all source points in S, respectively. With each obstacle B, we associate a height-balanced binary search tree (BST), like red-black tree, $T(B)$ whose leaves are designed to store (1) the vertical cuts touching the relative interior of the bottom edge of B; (2) $D_x(C, S)$ for each vertical cut C. Each internal node v of $T(B)$ stores the cut having the minimum sum of distances among cuts stored in the leaves of the subtree rooted at v.

We traverse HPG from the vertex corresponding to the top strip of P in a similar way to a depth first search. The differences are: (1) At each vertex of HPG adjacent to a pair of splitting edges, traverse the left edge first. (2) When we visit a vertex v through a left merging edge from a vertex u, stop traversing at v and back to u.

While traversing each vertex v of HPG, we construct a BST $T(v)$ whose leaves store the vertical cuts that pass through the corresponding horizontal strip of v in P. Suppose now that we are visiting an edge e of HPG first time from a vertex u to a vertex v, we do the following: Let C_e be the corresponding sub-cut of e in P.

(1) When e is a crossing edge, compute a candidate median point p on C_e. The point p is an intersection of C_e and the vertical cut $C(T(u))$ stored in the root of $T(u)$. Then $D(p) = D_y(C_e, S) + D_x(C(T(u)), S)$. After the computation, we make $T(u)$ to be the tree $T(v)$ of v.
(2) When e is a left splitting edge, we split $T(u)$ into three trees T_l, T_m, T_r, where T_m keeps the cuts in $T(u)$ that touch the relative interior of top edge of B. T_l (T_r) keeps the cuts in $T(u)$ to the left (right) of B, respectively. Let C_l, C_m, C_r be the left, middle, right sub-cuts of a cut of an obstacle such that $C_l = C_e$. We compute candidate median points on C_l, C_m, C_r from cuts in T_l, T_m, T_r, respectively, as in case (1). We leave T_r at vertex u so that it will be used when the right splitting edge adjacent to u is traversed. After the computation, we make T_l to be the tree, $T(v)$, of v.

(3) When e is a right splitting edge, we make the tree left at u while traversing the left splitting edge of u to be the tree $T(v)$ of v.
(4) When e is a left merging edge, suppose C_e is the left sub-cut of an obstacle B. We compute a candidate median point on C_e from the cut in the root of $T(u)$ and compute a candidate median point on the bottom edge of B as in case (1). After the computation, we merge $T(u)$ with the tree $T(B)$ associated with B.
(5) When e is a right merging edge, do in a symmetric way as case (4).

Then a median point is the point among candidate points on horizontal cuts that has the minimum sum of distances to all source points.

Theorem 2. *Given a set of m source points and a set of n rectangle obstacles, a median point can be computed in $O(m(m+n)\log n)$ time.*

3.3 Computing the Median Set

In this subsection, we present an algorithm for computing median set of source points S. In general, by Theorem 1, a median set is a set of rectilinear polygons, called *median polygon*, that may have holes in each polygon. In a degenerate case, polygons reduce to isolated points or line-segments.

Each median polygon P can be partitioned into a set of vertical strips separated by line-segments extending vertical edges of P and edges of holes on P. As an output of our algorithm, each median polygon is represented by such a set of vertical strips.

Consider all sub-cuts of obstacles and source points. For a horizontal sub-cut C, let $C_{+\epsilon}$, $C_{-\epsilon}$ be horizontal line-segments of the same length as C and lying above, below C apart from C by $\epsilon > 0$, respectively. Similarly, for a vertical sub-cut C, let $C_{+\epsilon}$, $C_{-\epsilon}$ be vertical line-segments to the right, left of C, respectively. For a horizontal sub-cut C, C is called a *plane-above* sub-cut if $D_y(C,S) = D_y(C_{+\epsilon},S)$. If C is a plane-above sub-cut then, by the symmetry of Theorem 1 (i), all points in the subregion R of VP(S) bounded below by C has the same y-distances to source points. Similarly we define a *plane-below, plane-left, plane-right* sub-cut.

For each sub-cut C, we check whether it is a valley sub-cut or ridge sub-cut, or null sub-cut. If C is a valley sub-cut, we also check whether it is a plane-above, plane-below, plane-left, plane-right sub-cut. Note that a valley sub-cut cannot be both plane-above and plane-below, or both plane-left and plane-right sub-cut by Theorem 1 (ii).

The basic structure of our algorithm is the same as the algorithm computing one median point. We construct HPG of the free space. We associate a BST $T(u)$ with each vertex u of HPG. Leaf nodes of BST store the information on the following objects that may constitute median regions: (1) A set of vertical line-segments which are subsets of vertical sub-cuts intersecting the horizontal strip $R(u)$ corresponding to the vertex u; (2) A set of rectangular subregions of $R(u)$ bounded by two consecutive vertical line-segments in (1). If the object is a vertical line-segment C, then the x-coordinate of C and $D_x(C,S)$ are stored in the leaf node. If the object is a rectangle R bounded by vertical line-segments C_1, C_2, then an interval $[x_1, x_2]$ is stored, where x_i is the x-coordinate of C_i, for $1 \leq i \leq 2$. If C_1 is a subset of a plane-right sub-cut or C_2 is a subset of a plane-left sub-cut, then all points in the rectangle R could be median points. (Thus we store $D_x(C_1, S)$ in the leaf node.) Otherwise, points inside of R cannot be median points, therefore the value ∞ is stored in the leaf node. Furthermore in each leaf node v, the y-coordinate of the horizontal cut that bounds above median regions containing the vertical line-segments or rectangles represented by v is stored in the leaf node.

BST has a secondary structure to support searching all leaf nodes that have the minimum sum of x-distances. By the secondary structure, we can search the leaf nodes in $O(K)$ time, where K is the number of the searched leaf nodes. Furthermore we can merge two BST's, and split a BST into two BST's in $O(\log N)$ time, where N is the size of BST.

From Theorem 1, median regions are bounded by valley sub-cuts. Thus we check whether the corresponding sub-cuts of edges of HPG bound median regions while traversing HPG. Leaf nodes of BST have the information on the x-coordinates of vertical strips or vertical line-segments which constitute median regions. The y-coordinates of edges bounding the vertical strips or vertical line-segments will be decided while traversing HPG. Suppose that we have the optimal value D^* of Equation 1 which is computed by the algorithm in previous section. While traversing each edge e of HPG first time from a vertex u to a vertex v, if e is a crossing edge, we do the following: Let C_e be the corresponding sub-cut of e in VP(S) and $D_x(T(u))$ be the minimum sum of x-distances of leaf nodes in $T(u)$. If $D_x(T(u)) + D_y(C_e, S) > D^*$, then no median point lies on C_e. Thus we do nothing. Otherwise, there are median points on C_e and we do the following: Note that C_e cannot be a ridge sub-cut.

(1) If C_e is a null sub-cut, then C_e does not bound below any median region by Theorem 1 (ii). Thus we do nothing for C_e.
(2) If C_e is a valley sub-cut, then there are median points on C_e. Furthermore if C_e is a plane-above or plane-below sub-cut, then C_e bounds median regions.
  (2.1) If C_e is a plane-below sub-cut, then several valley-subregions of VP(S) bounded above by C_e are median regions. Thus we set y-coordinates of leaf nodes having minimum sum of x-distances with the y-coordinate of C_e.
  (2.2) If C_e is a plane-above sub-cut, then several valley-subregions bounded below by C_e are median regions. Thus a set of vertical strips or vertical line-segments are generated as an output. Suppose that the object represented by a leaf node r of $T(u)$ is a candidate median region and the leaf node has a y-coordinate r_y. If r represents a vertical line-segment which is subset of some vertical sub-cut C, then a median region created by r is a vertical line-segment $[(C)_x, (C)_x] \times [(C_e)_y, r_y]$. If r represents a rectangle, then a median region created by r is a vertical strip $[x_i, x_{i+1}] \times [(C_e)_y, r_y]$, where the interval $[x_i, x_{i+1}]$ is stored for the rectangle in the leaf node.

(2.3) Otherwise, the median set lies on C_e. The outputs are horizontal line-segments or isolated points on C_e.

If e is a splitting or merging edge, then we do the similar work as we have done when we traverse a crossing-edge; for details, refer to the full version of this paper.

For binary search trees associated with obstacles and the top strip of VP(S), we merge consecutive leaf nodes into one leaf node if they have the same sum of x-distances. Furthermore when two BST's T_1, T_2 are merged, the rightmost interval leaf node of T_1 and and the leftmost interval leaf node of T_2 will be merged only if the intervals do not constitute median set. By merging these consecutive leaf nodes, we can reduce the number of vertical strips or line-segments which will be output of our algorithm to $O(K)$, where K is the number of vertices of median set. Thus we have:

Theorem 3. *Given a set of m source points and a set of n rectangle obstacles, the median set can be computed in $O(m(m + n) \log n + k)$ time, where k is the complexity of the median set of S.*

4 Computing the Center Set in the Plane

Let S be a set of m source points each with a positive weight, and $\mathcal{B}$ be a collection of n pairwise-disjoint axis-parallel rectangle obstacles. In this section, the center set of S with respect to $\mathcal{B}$ will be computed. Due to the space limitation, we will explain only the outline of the algorithm.

Before explaining our algorithm, we analyze the worst-case complexity of the center set. Its lower bound directly came from Figure 1 (b), and its upper bound was proved by using some properties about valleys and ridges of S in the presence of $\mathcal{B}$.

Lemma 2. *The worst-case complexity of a center set of S with respect to $\mathcal{B}$ is $\Theta(nm)$, where $|\mathcal{B}| = n$ and $|S| = m$.*

Note that the center set is a collection of line segments (possibly, isolated points) each of which has a slope of 45 or 135 degrees with respect to X-axis.

Our algorithm for computing the weighted center of the source points in the presence of the obstacles briefly works as follows: (i) draw cuts of the obstacles, partitioning the free space of FP($\mathcal{B}$) into $O(n^2)$ rectangular regions; (ii) for each rectangular region R compute the center set (and its radius) of S with respect to $\mathcal{B}$ when the center set is restricted to locate in R only; and finally, (iii) collect the centers with minimum radius.

Step (i) Draw cuts of all edges of the obstacles in $\mathcal{B}$. This can be done in $O(n \log n)$ time by vertical (horizontal) trapezoidal decomposition. This results in $O(n^2)$ *rectangular regions* and intersection points, called *grid points*. We compute the distances of the shortest paths between all grid points and all source points. Consider a grid point g and a source point s. Construct the monotone region and pyramids (including valleys and ridges) relative to s. While constructing $(\pm Y)$- (resp., $(\pm X)$-) pyramids, compute the x- (resp., y-) distances to s of vertical (resp., horizontal) cuts and ridges of s in the pyramids. This needs $O(mn^2 + mn \log n)$ time since there are $O(m)$ pyramids each taken $O(n \log n)$ construction time, and $O(mn^2)$ pairs of grid points and source points each taken $O(1)$ time for computing shortest distance between them.

Step (ii) Given a fixed rectangular region R, we want to compute the center set (and its radius d_R^*) of S in the presence of $\mathcal{B}$ when the center set is restricted to R. For this, we employ the *parametric search* [9]. Parametric search is an optimization technique which can be applied in situations where we seek a minimum parameter d^* satisfying the *monotone condition* that is met by all $d \geq d^*$ but not by any $d < d^*$. The strategy of the parametric search is to design sequential and parallel algorithms for the corresponding decision problem: given a parameter d, decide whether $d < d^*$ or $d \geq d^*$. If the sequential algorithm runs in T_s time and the parallel algorithm runs in T_p time using W_p processors, the parametric search results in a sequential algorithm with running time $O(W_p T_p + T_s T_p \log W_p)$ for computing d^*.

In our problem d_R^* satisfies the monotone condition, so parametric search can be used to compute d_R^*. Parametric search requires sequential and parallel algorithms for the decision problem: Given a value d, decide whether there is a point c in R such that $\max_{s \in S} d(c, s) \leq d$ (i.e., whether $d \geq d_R^*$).

For a source point s, let $\mathcal{R}_s(d)$ be the region of $\mathrm{FP}(\mathcal{B})$ that is reachable from s with distance d or less in the presence of the obstacles. Define $R_s(d) = R \cap \mathcal{R}_s(d)$ Then, $d \geq d_R^*$ if and only if $\cap_{s \in S} R_s(d) \neq \emptyset$. If there is no obstacle in the plane, $\mathcal{R}_s(d)$ is an L_1-metric disk with radius d, which is an axis-parallel square with diagonal length $2d$ rotated 90 degrees. Since obstacles make paths detour, $\mathcal{R}_s(d)$ in the presence of the obstacles is a subset of that in the absence of the obstacles.

Consider a region R and a source point s. $R_s(d)$ is defined as a "wedge" whose boundaries are two half-rays with slope of 45 and 135 degrees, respectively. Since we have computed the distances between all grid points and all source points, we can easily decide the shape of $R_s(d)$ (wedge) for each source point s. Now we need to $\bigcap R_s(d)$. For this, we use the fact the boundary of $R_s(d)$ consists of only two kinds of half-rays; slopes of 45 and 135 degrees. After all, we can compute $\bigcap R_s(d)$ in $O(m)$ sequential-time and in $O(\log m)$ parallel-time using $O(m)$ processors. Putting these into parametric search mechanism gives $O(m \log^2 m)$ time algorithm for computing the center set (and its radius d_R^*) inside R.

Step (iii) Since there are $O(n^2)$ rectangular regions, in $O(mn^2 \log^2 m)$ time we can find d^* and collect the centers achieving it.

Theorem 4. *Given a set of m weighted source points and a set of n rectangle obstacles, the center set can be computed in $O(mn^2 \log^2 m)$.*

For unweighted source points, we do not rely on parametric search but can develop a rather simple, direct algorithm.

Theorem 5. *Given a set of m unweighted source points and a set of n rectangle obstacles, the center set can be computed in $O(mn^2)$.*

The complexity of the above algorithms seems to be far from the worst-case complexity of the center set, i.e., $O(nm)$. The main difference between median and center problems is that whereas at least one point on valleys should be a median point, no points of valleys could be a center point. We believe that such a difference makes the center problem harder than the median one.

References

1. M. J. Atallah, R. Cole, and M. T. Goodrich, Cascading divide-and-conquer: a technique for designing parallel algorithms, *SIAM J. Comput.*, 18:499–532, 1989.
2. V. Chepoi and F. Dragan, Computing a median point of a simple rectilinear polygon, *Inform. Process. Lett.*, 49:281-285, 1994. 30
3. M. E. Dyer, On a multidimensional search technique and its application to the Euclidean one-centre problem, *SIAM J. Comput.*, 15:725–738, 1986. 29
4. P. J. de Rezende, D. T. Lee, and Y. F. Wu, Rectilinear shortest paths in the presence of rectilinear barriers, *Discrete Comput. Geom.*, 4:41-53, 1989. 31
5. R. L. Francis and J. A. White, *Facility Layout and Location*, Prentice Hall, Englewood Cliffs, NJ, 1974. 29
6. M. T. Ko and R. C. T. Lee, On weighted rectilinear 2-center and 3-center problems, *Inform. Sci.*, 54:169–190, 1991. 30
7. Y. Kusakari and T. Nishizeki, An algorithm for finding a region with the minimum total L_1-distance from prescribed terminals, *Proc. of ISAAC'97, Lecture Notes in Computer Science*, Springer-Verlag, 324–333, 1997. 30
8. R. C. Larson and G. Sadiq, Facility locations with the Manhattan metric in the presence of barriers to travel, *Oper. Res.*, 31:652–669, 1983. 30
9. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM*, 30:852–865, 1983. 39
10. M. Sharir and E. Welzl, Rectilinear and polygonal p-piercing and p-center problems, *Proc. 12th ACM Symp. Comput. Geom.*, 122–132, 1996. 30

Maximizing Agreement with a Classification by Bounded or Unbounded Number of Associated Words

Hiroki Arimura[1] and Shinichi Shimozono[2]

[1] Dept. of Informatics, Kyushu Univ., Fukuoka 812-8581, Japan
arim@i.kyushu-u.ac.jp
[2] Dept. of Artificial Intelligence, Kyushu Inst. of Tech., Iizuka 820–8502, Japan
sin@ai.kyutech.ac.jp

Abstract. We study the efficient discovery of word-association patterns, defined by a sequence of strings and a proximity gap, from a collection of texts with binary labels. We present an algorithm that finds all d strings and k proximity word-association patterns that maximizes agreement with the labels. It runs in expected time complexity $O(k^{d-1}n\log^{d+1} n)$ and $O(k^{d-1}n)$ space with the total length n of texts, if texts are uniformly random strings. We also show that the problem to find a best word-association pattern with arbitrarily many strings is MAX SNP-hard.

1 Introduction

Data mining emerged in early 1990's [1] aims to devise semi-automatic tools for discovering valuable "association rules" from facts stored in large scale databases. An association rule is an implication between a presumptive condition and an objective condition [6] such that if a record (tuple of attribute values or items) in a database satisfies the presumptive condition then with high probability the record also satisfies the objective condition. To find all qualified rules, a data mining tool receives a positive integer *support* and a ratio *confidence* from users, and searches for any association rule such that the number of records satisfying the rule is more than the support and the ratio of this number to the number of records satisfying the presumptive condition is larger than the confidence.

A considerable amount of results has been known for mainly well-defined and structured databases, such as relational databases with boolean or numeric attributes (e.g. [1,6]). Beside this, recent progress of measuring and sensing technologies and network infrastructure has been rapidly increasing the size and the species of weakly-structured databases, such as text databases and SGML/HTML archives. These lead to potential demands for data mining tools for databases where attributes or data structures may not be assumed in advance. However, there are still a few results [3,14] in this research direction. One difficulty in realizing them might be that a tool should not only process quickly to find rules but also construct attributes or structures defining rules in runtime.

Our aim is to develop an efficient tool for text databases along the lines of data mining. We introduce the framework of optimal pattern discovery, where we concentrate to find presumptive conditions that form association rules with a given objective condition achieving the maximum confidence. This captures frequently appearing situations in data mining research [6] where the objective condition is specified in advance, or so simple. The problem is, given a set of texts and its binary labeling function, to find a condition whose classification agrees with that of the function for the maximum number of texts. We call this *maximum agreement problem*. The complementary version of this problem, the minimum disagreement problem, is tied with the computational learning theory in the framework of agnostic PAC-learnability [5,8].

As a condition over texts, we consider a *word-association pattern* $(p_1, \ldots, p_d; k)$ formed from a sequence of strings $p_1, \ldots, p_d$ and a positive integer gap k. It matches a text if every string occurs in the text in the given order proximate within the specified distance k. This extends traditional queries called *proximity patterns* consisting of two strings and a gap distance between them. An algorithm efficiently solving the maximum agreement problem for word-association patterns can be applied in a wide range of practical problems, such as finding a consensus motif from protein sequences in the computational biology researches.

By a naive algorithm, the maximum agreement problem of word-association patterns formed from at most d strings is solvable in $O(n^{2d+1})$ time with the total length n of given texts. However, the practical importance of the problem requests more efficient algorithms. Nakanishi *et al.* [12] showed that if $d = 1$ and no disagreement is allowed then a pattern can be found in linear time. Arimura *et al.* [3] devised the algorithm that finds a word-association pattern whose agreement is maximum in $O(m \cdot n^2)$ time for $d = 2$ where m is the number of texts. In this paper, we present an algorithm that for every $d \geq 1$ finds a d-words k-proximity word-association pattern with the maximum agreement in expected running time $O(k^{d-1} n \log^{d+1} n)$ and $O(k^{d-1} n)$ space, if texts are uniformly random strings. Even in the worst case the algorithm runs in time $O(k^d n^{d+1} \log n)$. These suggest that, for the inputs almost being random strings such as the biological sequences, the algorithm runs drastically faster.

On the other hand, if the number of strings in word-association patterns is not limited, the maximum agreement problem is intractable. Dealing with this hardness, we should look for a polynomial-time algorithm that solves the problem approximately with some guaranteed approximation ratio. We partially clarify this issue by presenting the nonapproximability of our maximization problem. We show that the maximum agreement problem of word-association patterns formed from arbitrary many words is hard to approximate within a factor arbitrarily close to one in polynomial time, i.e., has no polynomial-time approximation scheme (PTAS) unless P=NP. It is an interesting contrast that the problem is approximable in polynomial-time within a fixed approximation ratio.

The remainder of this paper is organized as follows. First, we introduce some notions and definitions. Next, we present the efficient algorithm that discovers all the best word-association patterns, and analyze its running time. Then we prove the nonapproximability of the problem. We end with some discussion on the relation between our problem and computational learning theory.

2 Notions and Definitions

The set Σ is a finite alphabet throughout this paper. We always assume some fixed total ordering on the letters in Σ. We refer by ϵ to the empty string in Σ^* whose length is zero. For a string $s \in \Sigma^*$, we denote by $s[i]$ with $1 \leq i \leq |s|$ the ith letter of s and by $|s|$ the length of s. The concatenation of two strings s and t in Σ^* is denoted by $s \cdot t$, or simply by st. If for a string t there exist (possibly empty) strings $u, v, w \in \Sigma^*$ such that $t = uvw$ then we say that u, v and w are a *prefix*, a *subword* (a *substring*) and a *suffix* of t, respectively. Let t be a substring of $s \in \Sigma^*$. An *occurrence of t in s* is a positive integer label $1 \leq i \leq |s|$ that specifies a position and a region where in s the substring t occurs, that is, an index i such that $s[i] \cdots s[i + |t| - 1] = t$.

Let ∞ be the symbol of the positive infinity. A *proximity word-association pattern π over Σ* (*word-association pattern*) is a pair $\pi = (p_1, \ldots, p_d\,; k)$ of a finite sequence of strings in Σ^* and a parameter k called *proximity* which is either a nonnegative integer or ∞. A proximity word-association pattern π *matches* a string $s \in \Sigma^*$ if there exists a sequence $i_1, \ldots, i_d$ of integers such that every p_j in π occurs in s at the position i_j and $0 \leq i_{j+1} - i_j \leq k$ for all $1 \leq j < l$. The notion (d, k)*-pattern* refers to a d-words k-proximity word-association pattern $(p_1, \ldots, p_d; k)$. If the proximity k is ∞, then we simply write $\pi = (p_1, \ldots, p_d)$ by omitting k. The concatenation of two patterns $\pi = (\alpha; k)$ and $\tau = (\beta; k)$ is defined by $\pi\tau = (\alpha\beta; k)$.

A *sample* is a finite set $S = \{s_1, \ldots, s_m\}$ of strings in Σ^* and an *objective condition over S* is a binary labeling function $\xi : S \to \{0, 1\}$. Each string s_i in S is a *document* and $\xi(s_i)$ is a *label*. Let S be a sample and ξ an objective condition associated with S. Then, for a string $s \in S$, we say that a word-association pattern π *agrees with* ξ on s if π matches s if and only if $\xi(s) = 1$. The notion $|S|$ refers to the number of elements in S. For nonnegative integers i, j with $i \leq j$, we denote by $[i..j]$ the interval $\{i, i+1, \ldots, j\}$. We define $[i..j]$ to be empty if $j < i$. A *d-dimensional axis-parallel rectangle* or *d-dimensional box* is a region $[i_1..j_1] \times \cdots \times [i_d..j_d]$ with integers $i_1, j_1, \ldots, i_d, j_d$.

Definition 1. MAX AGREEMENT BY d-WORDS k-PROXIMITY ASSOCIATION
An instance is a tuple (Σ, S, ξ, d, k) of an arbitrary large alphabet Σ, a set $S \subseteq \Sigma^*$ of documents, an objective condition $\xi : S \to \{0, 1\}$ and positive integers d and k. A solution is a d-words k-proximity pattern π over Σ, and the measure of π is the number of documents in S on which π agrees with ξ. The goal of the problem is to find a word-association pattern π maximizing the measure.

We refer to the enhanced problem where the parameters d and k are not restricted by simply MAX AGREEMENT BY WORD ASSOCIATION.

3 Computing the Maximum Agreement by Association of a Bounded Number of Words

In this section, we describe an efficient algorithm that solves the maximum agreement problem for (d, k)-proximity patterns in expected time $O(k^{d-1} n \log^{d+1} n)$ and space $O(k^{d-1} n)$ when (1) S is a set of uniformly random strings with total length n over a finite alphabet, and (2) d and k are constants.

Before looking inside of the algorithm, let us briefly review a key data structure, a *suffix tree* [10]. Let t be a text of length n, and for $1 \leq p \leq n$ let t_p be the suffix of t starting at the position p. The *suffix tree for t* is a rooted tree $Tree_t$ that satisfies the following: (i) Each edge is labeled by a subword w of t, with which any other edge from the same node must be distinguished by the first letter of their labels; (iii) Each node v represents a subword $Word(v)$ of t obtained by concatenating the labels on the path from the root to v in its order.

Let s be a subword of a string t. The *locus* of s in $Tree_t$, denoted by $Locus(s)$, is the unique node v of $Tree_t$ such that s is a prefix of $Word(v)$ and $Word(w)$ is a proper prefix of s, where w is the parent of v. The subword w on an edge of $Tree_t$ is referred by the occurrence of w in t.

It is clear that $Tree_t$ has exactly n leaves and at most $n-1$ internal nodes, and can be represented in $O(n)$ space. McCreight [10] gives an elegant algorithm that computes $Tree_t$ in linear time and space. The *height* of $Tree_t$ is the maximum number of nodes on a path from the root to a leaf. The expected height of suffix trees is known to be very small for a random string [4].

By assuming all edges from the same node are lexicographically ordered, the ith leaf l_i for $1 \leq i \leq n$ in some appropriate traversal represents the suffix of rank i in the lexicographic order. Let $A_{p_1}, A_{p_2}, \ldots, A_{p_n}$ be the sequence of all the suffices of A in lexicographic order. Then, we store the indices $p_1, p_2, \ldots, p_n$ in an array suf of length n in this order, and define *pos* to be the inverse array of suf of length n, that is, $pos(suf(x)) = x$ for all index $1 \leq x \leq n$. These arrays are called the *suffix arrays* [9].

Let (Σ, S, ξ, d, k) be an instance of the maximum agreement problem. Obviously, the maximization of the agreement equals the maximization of the measure $\Delta_{S,\xi}(\pi) = count_1 – count_0$, where $count_\alpha = |\{ s \in S \mid \text{“}\pi \text{ matches } s\text{” and } \xi(s) = \alpha \}|$ for each $\alpha \in \{0, 1\}$. In Fig. 1, we show the algorithm.

Let $\$ \notin \Sigma$ be a special delimiter symbol such that $\$ \neq \$$. Given a sample $S = \{s_1, \ldots, s_m\}$,the algorithm builds a single string $A = s_1\$ \cdots \$s_m\$$ of length $n = |A|$ and the suffix tree $Tree_A$ for all strings padded in A in linear time. For $1 \leq p \leq n$, we define $\delta(p) = i$ if p belongs to the ith document $s_i \in S$. A pattern π is in *canonical form* if it can be expressed as $\pi = (Word(u_1) \ldots, Word(u_d); k)$ for a sequence of nodes $u_1, \ldots, u_d$ of $Tree_A$. Here, if u_j is a leaf then we define $Word(u_j)$ to be the longest prefix of the subword represented by u_j that does not contain \$. If u_j is an internal node then we can see that $Word(u_j)$ does not contain \$. This is because \$ cannot match any symbol including itself, and thus, every node corresponding to \$ must be a lowest branching node. Let $\perp_S \in \Sigma^*$ be an arbitrary string whose length is $\max\{\, |s| \mid s \in S\} + 1$. Clearly, $\Delta_{S,\xi}(\perp_S) = 0$.

```
Algorithm. Bounded_Maximum_Agreement:
Input: An alphabet $\Sigma$, an integer $k \geq 0$, a sample $S \subseteq \Sigma^*$ and an objective
condition $\xi : S \to \{0,1\}$.
Output: A $(d,k)$-proximity pattern $\pi$ over $\Sigma$ that maximizes $\Delta_{S,\xi}(\pi)$.
  1 Let $A = s_1\$\cdots\$s_m\$$ be the input text associated with $S$. Compute the
    suffix tree $Tree_A$ for $A$, the suffix arrays $suf$ and $pos = suf^{-1}$.
  2 Let $Diag_{d,k,S}$ be the $d$-dimensional diagonal set of width $k$. Then, com-
    pute $Rank_{d,k,S} = \{ (x_d, \ldots, x_1; \delta) \mid x_j = pos(i_j)$ for every $1 \leq j \leq$
    $d$ and $(i_d, \ldots, i_1; \delta) \in Diag_{d,k,S} \}$ by transforming the labeled points in
    the position space $Diag_{d,k,S}$ into those in the rank space $Rank_{d,k,S}$ using
    the suffix array $pos$.
  3 Call $Discover(d, Rank_{d,k,S}, \rho_d)$ with the empty pattern $\rho_d := (;k)$.
  4 Return the best patterns $\pi$ found if the maximum of $\Delta$ is positive. Other-
    wise, return $\perp_S$.
```

Fig. 1. The algorithm for computing the maximum agreement over (d,k)-proximity patterns

Lemma 1. *$\Delta_{S,\xi}(\pi)$ is maximized over all (d,k)-proximity patterns by either a pattern π in canonical form or $\perp_S$.*

This can be shown by the discussion similar to that for $d = 2$ in [3].

By Lemma 1, it is sufficient to consider only the patterns in canonical form. The idea is that to search and test such patterns, we reduce the search of d-words patterns to that of d-dimensional boxes over the space formed by the lexicographically ordered suffices. This allows an average case efficient algorithm with random inputs by visiting only a small fraction $O(h^d N)$ of patterns from $O(n^d)$ possible combinations of d substrings of A, where h is the height of the suffix tree for the input text and $N = k^{d-1}n$ is the number of points in $Diag_{d,k,S}$.

The *d-dimensional diagonal set of width k*, denoted by $Diag_{d,k,S} \subseteq [1..n]^d \times [1..m]$, is the set of all the labeled d-dimensional points $(i_1, \ldots, i_d; \delta)$ such that for $i_0 = 0, i_{d+1} = n$ and some $0 \leq \delta \leq m$, every i_j with $1 \leq j \leq d$ satisfies $0 \leq i_{j+1} - i_j \leq k$ and $\delta(i_j) = \delta$. Let $Rank_{d,k,S} \subseteq [1..n]^d \times [1..m]$ be the set of all the labeled d-dimensional points $(x_d, \ldots, x_1; \delta)$ such that $(i_d, \ldots, i_1; \delta) \in Diag_{d,k,S}$ and $x_j = pos(i_j)$ for every $1 \leq j \leq d$. We call $Diag_{d,k,S}$ and $Rank_{d,k,S}$ the position and the rank spaces.

We can see that for any node v of the suffix tree $Tree_A$, the set of the occurrences of $Word(v)$ form a consecutive subinterval $I(v) = [x_1..x_2]$ of the rank space $[1..n]$. Then, we relate each (d,k)-word association pattern $\pi = (Word(v_1), \ldots, Word(v_d); k)$ in canonical form with the d-dimensional box $Box(\pi) = I(v_1) \times \cdots \times I(v_d)$ over $Rank_{d,k,S}$. For any set $Q \subseteq [1..n]^d \times [1..m]$, we define the measure of box $B(\pi)$ by $\Delta_{Q,\xi}(B(\pi)) = count'_1 - count'_0$, where $count'_\alpha$ is the number of the indices $\delta \in [1..m]$ such that $(x_1, \ldots, x_d; \delta) \in Q \cap (Box(\pi) \times [1..m])$ and $\xi(\delta) = \alpha$ for each $\alpha \in \{0,1\}$. The following lemma is essentially due to Manber and Baeza-Yates [9].

```
Procedure. Discover($d, Q, \rho_d$):
  1. The case that $d \geq 1$:
     (a) Compute the set $X_{\rm coord}$ of the first coordinates of the points in $Q$ and
         sort it. Let $H := \emptyset$ and $H_{\rm new} := \emptyset$.
     (b) For each coordinate $x \in X_{\rm coord}$ do:
           Attach the set $Q_{\lambda_x} = \{ (x_{d-1}, \ldots, x_1; \delta) \mid (x_d, x_{d-1}, \ldots, x_1; \delta) \in Q \}$
           to the leaf $\lambda_x$ of rank $x$, and then append $\lambda_x$ to the right end of $H$.
     (c) For each level $l = 0, 1, \ldots, h$ do:
         (i) Scanning $H$ from left to right, for each $v \in H$ do:
               Let $w$ be the parent of $v$. Then, update $Q_w$ by $Q_w := Q_w \cup Q_v$
               in sorted order in the $(d-1)$st coordinate (in $\delta$ if $d = 1$). If $v$ is
               the leftmost child of $w$ then append $w$ to the right end of $H_{\rm new}$.
               If $l \geq 0$, then recursively call $Discover(d-1, Q_v, \rho_{d-1})$ with
               $\rho_{d-1} := \rho_d \cdot (Word(v); k)$.
        (ii) Discard $Q_v$ for all nodes $v \in H$. Let $H := H_{\rm new}$ and $H_{\rm new} := \emptyset$.
  2. The case that $d = 0$:
         Compute $count_\alpha := |\{\delta \in Q \mid \xi(\delta) = \alpha\}|$ for each $\alpha \in \{0, 1\}$. If $\Delta :=$
         $count_1 - count_0$ is larger than the largest value seen so far, record the
         pattern $\rho_0$.
```

Fig. 2. The procedure *Discover*

Lemma 2. *Let S be a sample and $Q = Rank_{d,k,S}$ be the set defined above. Then, $\Delta_{S,\xi}(\pi) = \Delta_{Q,\xi}(Box(\pi))$ for any (d,k)-pattern π in canonical form.*

Proof. Assume that a (d,k)-pattern $\pi = (p_1, \ldots, p_d; k)$ matches some document $s_\delta \in S$. By definition, we can see that $(i_1, \ldots, i_d) \in [1..n]^d$ is the vector of the occurrences of the components $p_1, \ldots, p_d$ of π if and only if $(i_1, \ldots, i_d; \delta) \in Diag_{d,k,S}$ and for every $1 \leq j \leq d$, index i_j is an occurrence of p_j in A. Since i_j is an occurrence of p_j if and only if $pos(i_j) \in I(v_j)$ where $v_j = Locus(p_j)$, this is equivalent to that $(pos(i_1), \ldots, pos(i_d); \delta) \in I(v_1) \times \cdots \times I(v_d) \times [1..m] = Box(\pi) \times [1..m]$ provided that $(i_1, \ldots, i_d; \delta) \in Diag_{d,k,S}$. Thus, we know that π matches $s_\delta \in S$ if and only if $Rank_{d,k,S} \cap Box(\pi) \times [1..m]$ contains a point labeled by the document name δ. □

Fig. 2 shows the subprocedure *Discover* that recursively computes best boxes $\Delta_{Q,\xi}(Box(\pi))$ by making a series of orthogonal range queries directly over the suffix tree with dynamic programming.

Theorem 1. *Let Σ be an alphabet, d and k are fixed integers, S be a sample, and ξ be an objective function for S. Then our algorithm computes all the (d,k)-patterns in canonical form that maximizes the measure over (d,k)-patterns in time $O(k^{d-1}h^d n \log n)$ and space $O(k^{d-1}n)$ in the worst case, where n is the total length of strings in S and $h \leq n$ is the height of the suffix tree for A.*

Proof. Given a sample S, the algorithm computes A, $Tree_A$, $Diag_{d,k,S}$ and $Rank_{d,k,S}$. Let $Q_d = Rank_{d,k,S}$. By Lemma 2, the maximum agreement problem reduces to finding a sequence $(v_1, \ldots, v_d)$ of nodes of $Tree_A$ such that the d-dimensional box $B = I(v_1)\times\cdots\times I(v_d)$ maximizes $\Delta_{Q_d,\xi}(B)$.

Let Q be any subspace of c-dimensional rank space for $c \le d$. In what follows, we write $(x_c, \ldots, x_1)$ for a c-dimensional point in Q and we denote by $Q|_{P(x_c,\ldots,x_1)}$ the subspace of Q consisting of all points $(x_c, \ldots, x_1) \in Q$ that satisfies the condition $P(x_c, \ldots, x_1)$.

Initially, *Discover* is invoked with $(d, Q_d, (;k))$. When *Discover* is invoked with (c, Q, ρ) for some $c \le d$, the procedure computes the subset Q_v using dynamic programming by visiting each node v from the leaves to the root. Then by induction on the height of v, Q_v is exactly the subset of Q consisting of the points with the c-th coordinate x_c is in $I(v)$, i.e., $Q_v = Q|_{x_c\in I(v)}$.

Let Q_c be any c-dimensional rank space and $B_c = I_c\times\cdots\times I_1$ be any box over Q_c, and let Q_{c-1} and $B_{c-1} = I_{c-1}\times\cdots\times I_1$ be their projections to $(c-1)$-dimensional rank space. Then, if $x_c \in I(v)$ for all $(x_c, \ldots, x_1) \in Q_v$ then the following equivalence holds: $\Delta_{Q_c,\xi}(B_c) = \Delta_{Q_{c-1},\xi}(B_{c-1})$. Thus, we can reduce the problem to that in the lower dimension in this way.

From these observations, it is not hard to see that whenever *Discover* is invoked with (c, Q_c, ρ_c) for $\rho_c = (v_d, \ldots, v_{c+1})$, Q_c equals $Q_d|_{x_d\in I(v_d),\ldots,x_{c+1}\in I(v_{c+1})}$. Thus, when the computation reaches $c = 0$, Q_0 becomes the set of the distinct labels in $Q_d \cap (I(v_d)\times\cdots\times I(v_1)\times[1..m])$, and Step 2 correctly computes $\Delta_{Q_d,\xi}(I(v_d)\times\ldots\times I(v_1))$. Hence, now we see that *Discover* computes the maximum of $\Delta_{S,\xi}$ by visiting all boxes that has nonempty intersection with $Rank_{d,k,S}$.

Next, we consider the time complexity of *Discover*. Let $N = k^{d-1}n$ be the cardinality of $Diag_{d,k,S}$ and, thus, of $Rank_{d,k,S}$. At Step 1 of the main algorithm, we compute $Tree_A$ and *pos*, *suf* in $O(n)$ time using the linear time algorithm of [10]. Step 2 is computable in $O(N)$. We assume each Q_v is implemented by an appropriate data structure that allows insertions in $O(\log n)$ time. At Step 3, the computation time $T_d(N)$ of $Discover(d, Q_d, \rho_d)$ with $N = |Q_d|$ is estimated as follows. If $d = 0$ then we can easily see $T_0(N) = O(N)$. Suppose that $d \ge 1$. Let H_l be the set of nodes in level l, and for each $v \in H_l$, let $N(v) = |Q_v|$. Then, we have the recurrence

$$T_d(N) = O(N) + \sum_{1\le l\le h}\sum_{v\in H_l} N(v)\log N(v) + \sum_{1\le l\le h}\sum_{v\in H_l} T_{d-1}(N(v))$$

with the constraint $\sum_{v\in H_l} N(v) = N$ for all l. The first term is for Step (a) and Step (b), the second term is for merging Q_v's and the third term is for computing $Discover(d-1, Q_v, \rho_{d-1})$. Solving this recurrence, we have $T_d(N) = O(dh^d N\log N)$. This completes the proof. □

Corollary 1. *Let S be a set of uniformly random strings with total length n over a finite alphabet. Then the problem* Maximum Agreement by d-Words k-Proximity Association *is solvable in expected time $O(dk^{d-1}n\log^{d+1} n)$ and space $O(k^{d-1}n)$, where (Σ, S, ξ, d, k) is an instance of the problem, and n is the total length of strings in S.*

Proof. Let A be a uniformly random string of b letters of length n over $\Sigma\cup\{\$\}$. Let H_n be the random variable that gives the height of the suffix tree for A of length n. For every $h \geq 0$, Devroye *et al.* [4] gives the bound $\mathbf{P}(H_n \leq h) \leq 2n(q^{h+1}/(1-q) + nq^{2h})$ on the probability that $H_n \leq h$, where $q = (1/b)^{1/2}$. Then, the average running time is $\mathbf{E}(T(n)) = \sum_A T(n,h) \cdot P(H_n = h)$, where the addition is taken over all A of length n and $T(n,h) = dk^{d-1}h^d n \log n)$. By calculating the addition separately for the cases $H_n \leq 2\log_b n$ and $H_n > 2\log_b n$, we have the expected running time $\mathbf{E}(T(n)) = O(dk^{d-1} n \log^{d+1} n)$. □

4 Hardness of Approximating Maximum Agreement by Association of Arbitrary Many Words

As we have seen, Max Agreement by d-Words k-Proximity Association for fixed d and k is solvable in polynomial time. However, the problem has seemed to be computationally intractable if d and k are not limited. This observation comes from the NP-completeness of a consistency problem of a class of regular patterns [11] in computational learning theory. Although the proof of this result does not capture the hardness of the optimization, it tells us that the maximum agreement problem of word-association patterns is intractable if the optimum is very close to the total number of texts.

Consider the following trivial algorithm: Given an instance, simply count the numbers of positive-labeled strings and negative-labeled strings and choose the better one from either an empty pattern accepting all labeled strings or a pattern rejecting all strings (for example a string longer than all given strings). This algorithm clearly approximates the problem with a guaranteed factor 1/2, and thus Max Agreement by Word Association is in class APX [2].

On the other hand, it is hard to approximate the problem within an arbitrarily small error in polynomial time. The main result in this section is the following theorem.

Theorem 2. Max Agreement by Word Association *is* Max SNP*-hard.*

Proof. We prove this by an L-reduction [13] from Max 2-Sat. We build for a Max 2-Sat instance $\phi = (X, C)$ of a set $X = \{x_1, \ldots, x_n\}$ of variables and a set $C = \{c_1, \ldots, c_m\}$ of 2-literal clauses, an instance $\pi = (A_\phi, T_\phi, \xi_\phi)$ of Max Agreement by Word Association.

We firstly define an alphabet $A_i = \{a_i, b_i\}$ for every $1 \leq i \leq n+1$ where n is the number of boolean variables $|X|$ in ϕ. Let $bin(i)$ for $1 \leq i \leq n$ be the binary representation of i in $(A_{n+1})^{\lceil \log n \rceil + 1}$ by some natural coding. We denote by symbols $\mathtt{t}_i, \mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i$ the strings $a_i a_i b_i, b_i a_i a_i, a_i b_i a_i, b_i a_i b_i$ in $(A_i)^3$ for all $1 \leq i \leq n$, respectively. Note that any word-association pattern that accepts both $\mathtt{d}_i$ and $\mathtt{u}_i$ cannot reject either $\mathtt{t}_i$ or $\mathtt{f}_i$, or both. Let S_i for $1 \leq i \leq n$ be a set either $\{\mathtt{t}_i, \mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i\}$, $\{\mathtt{t}_i, \mathtt{u}_i, \mathtt{d}_i\}$ or $\{\mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i\}$. Then the notion $[S_i]$ refers a pattern that accepts strings in S_i and rejects $\{\mathtt{t}_i, \mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i\} - S_i$. For example, $[\{\mathtt{t}_i, \mathtt{d}_i, \mathtt{u}_i\}]$ is a pattern $(a_i b_i)$ or (a_i, b_i), which accepts $\mathtt{t}_i, \mathtt{d}_i, \mathtt{u}$ but rejects $\mathtt{f}_i$.

Next, for each clause $c_i \in C$, we associate the following negative-labeled string: For each j from 1 to n, we concatenate either (i) $\mathtt{d}_j$ if x_j and $\bar{x}_j$ are not in c_i, (ii) $\mathtt{f}_j$ if the literal x_j appears in c_i, or (iii) $\mathtt{t}_j$ if $\bar{x}_j$ is in c_i. Then to its end we append $bin(i)$. For example, we associate with a clause $c_i = (x_j, \bar{x}_k)$ with $1 < j < k < n$ the string $\mathtt{d}_1 \cdots \mathtt{f}_j \cdots \mathtt{t}_k \cdots \mathtt{d}_n \cdot bin(i)$. Note that we are assuming no clause in C has two complementary literals. Such a clause can be handled as the auxiliary additional value to the measure in the reduction.

For each clause c_i, we associate two positive-labeled strings $\mathtt{d}_1 \cdots \mathtt{d}_n \cdot bin(i)$ and $\mathtt{u}_1 \cdots \mathtt{u}_n \cdot bin(i)$. We refer to these by *down-string* and *up-string*, respectively. The labeled sample corresponding to ϕ is defined as the set of the negative-labeled strings and the positive-labeled strings associated with all clause in C.

Now we specify the translation from a solution p for $(A_\phi, T_\phi, \xi_\phi)$ to a Boolean assignment B_p for ϕ. Let S_i for $1 \le i \le n$ be either $\{\mathtt{t}_i, \mathtt{u}_i, \mathtt{d}_i\}$ or $\{\mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i\}$. Then, a *boolean assignment-pattern* p *for* $(A_\phi, T_\phi, \xi_\phi)$ is a word-association pattern $[S_1] \cdot [S_2] \cdot \cdots \cdot [S_n] \cdot [A_{n+1}^{\lceil \log_2 n \rceil + 1}]$, where $[A_{n+1}^{\lceil \log_2 n \rceil + 1}]$ denotes a pattern that accepts all strings in $A_{n+1}^{\lceil \log_2 n \rceil + 1}$. By regarding $\{\mathtt{t}_i, \mathtt{u}_i, \mathtt{d}_i\}$ and $\{\mathtt{f}_i, \mathtt{u}_i, \mathtt{d}_i\}$ as 'true' and 'false' assignment to x_i respectively, an assignment pattern rejects a negative-labeled string if and only if the corresponding clause is satisfied. This fact is important since the following lemma holds:

Lemma 3. *For any solution p of $\pi = (A_\phi, T_\phi, \xi_\phi)$, we can compute in logspace an assignment-pattern p' for π whose measure is larger than that of p.*

We show a brief outline of a proof of this lemma. Without loss of generality, we can suppose that a solution p accepts both some up-strings and down-strings and achieves the measure larger than $2/3 \cdot |T_\phi|$. Otherwise we simply take the empty-pattern ( ) as p. Then, such a pattern p can be decomposed into the (possibly empty) subsequences p_i formed from only letters in A_i for all $1 \le i \le n+1$, since p must accept all the positive-labeled strings except those rejected by p_{n+1}.

If we modify the last segment $[p_{n+1}]$ to accept more positive-labeled strings, at most one negative-labeled string will be accepted for each pair of newly accepted up-string and down-string ending with $bin(i)$. Thus we can modify p_{n+1} to pattern that allows to accept all positive-labeled strings. Now let p_i be one of the subsequences of p accepting both $\mathtt{t}_i$ and $\mathtt{f}_i$. We replace every such subsequence p_i with either $[\{\mathtt{d}_i, \mathtt{u}_i, \mathtt{t}_i\}]$ or $[\{\mathtt{d}_i, \mathtt{u}_i, \mathtt{f}_i\}]$. Each choice can be done even arbitrarily, and every such replacement at least retains the number of rejected negative-labeled strings. Thus we obtain an assignment-pattern p' from p.

The Lemma 3 tells that the measure of the optimum solution of π is given by some assignment pattern, and a boolean assignment-pattern p corresponds to the boolean assignment, say B_p, to X with respect to the formula ϕ. Then, it is immediate that the following claim hold:

Claim. For every 2-CNF formula ϕ, there holds $opt(\pi) \le 5 \cdot opt(\phi)$. For every assignment-pattern p, there holds $opt(\phi) - m(\phi, B_p) = opt(\pi) - m(\pi, p)$.

The first statement is true because, for any 2-CNF formula ϕ, always $opt(\phi) \ge \frac{1}{2}|C|$ [7] and $opt(\pi) = 2 \cdot |C| + opt(\phi)$ hold. The second statement is clear since with an assignment-pattern all the positive-labeled strings are accepted and thus only the number of rejected negative-strings corresponding to the satisfied clauses makes a difference between $opt(\pi)$ and $m(\pi, p)$. □

Corollary 2. *There is no PTAS for* Max Agreement for Word Association Patterns *unless P = NP.*

5 Discussion

Agnostic PAC-learning [8] is a generalization of a well known PAC-learning model in computational learning theory, where the learner has no information about the structure behind the target function, or training examples may contain arbitrary large percent of noise. Kearns *et al.* [8] gives a characterization that for any hypothesis class of low *VC-dimension*, the polynomial time solvability and the efficient agnostic PAC-learnability for the class are equivalent. Although the problem is intractable in most cases, recently some geometrical patterns, e.g. *axis-parallel rectangles* and *convex k-gons* on Euclid plain, are shown to be efficiently agnostic PAC-learnable by this characterization [5]. Since word-association patterns obviously have polynomial VC-dimension, we know from Theorem 1 that our algorithm in Section 3 works as an efficient agnostic learning algorithm for d-words k-proximity word-association patterns. To our knowledge, this is one of the first results on the efficient agnostic learnability for nontrivial concept classes other than geometric patterns.

References

1. R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases. In Proc. the ACM SIGMOD Conference on Management of Data (1993), 207–216. 39
2. G. Ausiello, P. Crescenzi, M. Protasi, Approximate solution of NP optimization problems. *Theor. Comput. Sc.,* 150, 1–55 (1995). 46
3. H. Arimura, A. Wataki, R. Fujino, S. Arikawa, A fast algorithm for discovering optimal string patterns in large text databases. DOI-TR-148, Kyushu Univ. (To appear in Proc. the 9th Int. Workshop on Algorithmic Learning Theory) 39, 40, 43
4. L. Devroye, W. Szpankowski, B. Rais, A note on the height of the suffix trees. *SIAM J. Comput.*, 21, 48–53 (1992). 42, 46
5. D. P. Dobkin, D. Gunopulos, W. Maass, Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Comut. Sys. Sci.*, 52, 453–470 (1996). 40, 48
6. T. Fukuda, Y. Morimoto, S. Morishita, T. Tokuyama, Data mining using two-dimensional optimized association rules. In Proc. 1996 SIGMOD (1996), 13–23. 39, 40
7. D. S. Johnson, Approximation algorithms for combinatorial problems. *J. Comut. Sys. Sci.*, 9, 256–278 (1974). 47
8. M. J. Kearns, R. E. Shapire, L. M. Sellie, Toward efficient agnostic learning. *Machine Learning*, 17, 115–141, (1994). 40, 48
9. U. Manber, R. Baeza-Yates, An algorithm for string matching with a sequence of don't cares. *Inf. Procc. Lett.*, 37, 133–136 (1991). 42, 43
10. E. M. McCreight, A space-economical suffix tree construction algorithm. *J. ACM*, 23, 262–272 (1976). 42, 45

11. S. Miyano, A. Shinohara and T. Shinohara, Which classes of elementary formal systems are polynomial-time learnable. Proc. 2nd Workshop on Algorithmic Learning Theory, 139-150 (1991). 46
12. M. Nakanishi, M. Hashidume, M. Ito, A. Hashimoto, A linear-time algorithm for computing characteristic strings. In Proc. 5th ISAAC (1994), 315-323. 40
13. C. H. Papadimitriou, M. Yannakakis, Optimization, approximation, and complexity classes. *J. Comput. Sys. Sci.*, 43, 425–440 (1991). 46
14. J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, K. Zhang. Combinatorial pattern discovery for scientific data: some preliminary results. In Proc. 1994 SIGMOD (1994), 115–125. 39

Disjunctions of Horn Theories and Their Cores

Thomas Eiter[1], Toshihide Ibaraki[2], and Kazuhisa Makino[3]

[1] Institut für Informatik, Universität Gießen
Arndtstraße 2, D-35392 Gießen, Germany
eiter@informatik.uni-giessen.de

[2] Department of Applied Mathematics and Physics
Graduate School of Informatics, Kyoto University
Kyoto 606, Japan
ibaraki@kuamp.kyoto-u.ac.jp

[3] Department of Systems and Human Science
Graduate School of Engineering Science, Osaka University
Toyonaka, Osaka, 560, Japan
makino@sys.es.osaka-u.ac.jp

Abstract. In this paper, we study issues on disjunctions of propositional Horn theories. In particular, we consider deciding whether a disjunction of Horn theories is Horn, and, if not, computing a Horn core, i.e., a maximal Horn theory included in this disjunction. The problems are investigated for different representations of Horn theories, namely for Horn CNFs and characteristic models. While the problems are shown to be intractable in general, we present polynomial time algorithms for bounded disjunctions in the formula-based case.

1 Introduction

Since deduction from a set of propositional clauses is a well-known co-NP-complete problem, different approximation methods for reasoning from a clausal theory have been investigated, e.g., [10,13,3,4,1]. One of these approaches [10] uses a Horn greatest lower bound (also called *Horn core* [11]), i.e., a maximal Horn theory $\Sigma_c \subseteq \Sigma$, if we view theories as sets of models, and the least Horn upper bound (*Horn envelope* [11]), which is the minimal Horn theory $\Sigma_e \supseteq \Sigma$, i.e., $\Sigma_e \subseteq \Sigma'$ for any Horn theory Σ' such that $\Sigma' \supseteq \Sigma$. Computing Horn envelopes and Horn cores has been investigated in [10,11,2,4,8,6,1]. It has been shown that a Horn core of a given CNF formula φ is computable in polynomial time with an oracle for NP [2], and that computing a maximum (in terms of the numbers of models) Horn core is co-NP-hard [11], if the theory Σ is given by the set of its models.

In this paper, we consider the issue of computing Horn cores of the disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i, represented either by Horn CNFs, or by their *characteristic models* [9]. Characteristic models have been proposed as a model-based alternative to formula-based theory representation. The two approaches are orthogonal with respect to space requirements, i.e., the one approach sometimes allows for an exponentially smaller representation than the other; see [9,12]. Observe that a disjunction Σ of Horn theories is in general not Horn. Hence, it is of interest whether Σ is Horn, in particular, since then the Horn core and the Horn envelope coincide to Σ.

Disjunctions of Horn theories may be encountered in different applications. For example, suppose that two groups have respectively formed logical hypotheses about an application domain (the "world"), e.g., relationships between medical tests and diseases, and they believe the relationships amount to a Horn theory. The hypotheses Σ_1 and Σ_2 of two groups (amounting to sets of models) are obtained from actual and conjectured cases, respectively, i.e., concrete measurement data (obtained by experiments) and data which are believed to be true. Suppose the hypotheses Σ_1 and Σ_2 are merged. Then, at the logic level, the disjunction, i.e., union $\Sigma = \Sigma_1 \cup \Sigma_2$ describes the merged hypotheses. It is of particular interest to know whether Σ is Horn; if so, then the hypotheses are compatible in the sense that no further cases have to be adopted in order to preserve the Horn property, which may indicate that the individual hypotheses are sound. However, if Σ is not Horn, then either further cases have to be added to maintain the Horn property (which corresponds to derivation of new case knowledge), or some of the adopted hypothetical cases have to be abandoned. Applying Occam's razor, it is natural to add or to abandon a minimal set of cases. In the former case, this amounts to finding the Horn envelope of Σ, and in the latter case, to finding a Horn core Π of Σ. In finding a Horn core Π, it may be also asked to add a constraint such that Π includes all actual cases Σ_1, i.e., computing a Horn core Π satisfying $\Sigma_1 \subseteq \Pi \subseteq \Sigma$ can be seen as one of the interesting and important problems.

In this paper, we first address whether a disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ of Horn theories Σ_i, represented either by Horn CNFs or by their characteristic models, becomes a Horn theory. We show that checking this property is co-NP-complete for both representations in general, but is polynomially solvable if l is bounded by a constant. We next deal with the problem of computing a Horn core Π of Σ satisfying $\Sigma_1 \subseteq \Pi \subseteq \Sigma$, which has arisen in the above example. We show that, if l is bounded by some constant, then the problem is polynomially solvable from Horn CNFs, while it is co-NP-hard from characteristic models, even if $l = 2$. For the formula-based representation, we present a polynomial time algorithm CORE to compute a Horn core of the disjunction of two Horn theories Σ_1 and Σ_2. We also develop an algorithm CORE* for computing a Horn core of the disjunction of $l\,(\geq 3)$ Horn theories. This algorithm is polynomial if l is bounded by some constant. Furthermore, we give structural characterizations of Horn cores of a disjunction of Horn theories. As for the Horn envelope (which is always unique), we show that it is polynomially computable from characteristic models, but not from Horn CNFs, even if $l = 2$.

Due to space constraints, most proofs are omitted. They are given in [7].

2 Preliminaries

We assume a supply of propositional variables (atoms) $x_1, x_2, \ldots, x_n$, where each x_i evaluates to either 1 (true) or 0 (false). Negated variables are denoted by $\overline{x}_i$. These x_i and $\overline{x}_i$ are called literals. A clause is a disjunction $c = \ell_1 \vee \cdots \vee \ell_k$ of literals, while a term is a conjunction $\ell_1 \wedge \cdots \wedge \ell_k$ of literals. By $P(c)$ and $N(c)$ (resp., $P(t)$ and $N(t)$) we denote the sets of variables occurring positively and negatively in c (resp., t); $\perp$ (resp., $\top$) is the empty clause (resp., empty term), i.e, falsity (resp., truth). A conjunction of clauses $\varphi = \bigwedge_i c_i$ (resp., a disjunction of terms $\varphi = \bigvee_i t_i$) is called *conjunctive normal form* (CNF) (resp., *disjunctive normal form* (DNF)).

A *model* is a vector $v \in \{0,1\}^n$, whose i-th component is denoted by v_i. A *theory* is any set $\Sigma \subseteq \{0,1\}^n$ of models. We denote by $v \leq w$ the usual bitwise ordering of models, where $0 \leq 1$. A model $v \in \Sigma$ is *minimal* in Σ, if there is no $w \in \Sigma$ such that $w < v$. For $B \subseteq \{1, 2, \ldots, n\}$, we denote by x^B the model v such that $v_i = 1$, if $i \in B$ and $v_i = 0$, if $i \notin B$.

For any formula φ, let $T(\varphi) = \{v \in \{0,1\}^n \mid \varphi(v) = 1\}$ be its set of models. A theory Σ is represented by φ, if $T(\varphi) = \Sigma$. If unambiguous, we often do not distinguish a formula from the theory it represents. For any φ and ψ, we write $\varphi \leq \psi$ if $T(\varphi) \subseteq T(\psi)$. A nontautological clause c (resp., noncontradictory term t) is an *implicate* (resp., *implicant*) of a theory Σ if $c(v) = 1$ for all $v \in \Sigma$, (resp., $t(v) = 0$ for all $v \notin \Sigma$); it is *prime*, if no proper subclause (resp., subterm) is an implicate (resp., implicant) of Σ.

A theory Σ is *Horn* if $\Sigma = Cl_\wedge(\Sigma)$, where $Cl_\wedge(S)$ is the closure of $S \subseteq \{0,1\}^n$ under bitwise AND (i.e., intersection) of models v, w, denoted by $v \bigwedge w$. Any Horn Σ has the least (unique minimal) model, given by $\bigwedge_{v \in \Sigma} v$, where $\bigwedge_{w \in S} w$ for $S \subseteq \{0,1\}^n$ is the intersection of all models in S. A clause c is *Horn* if $|P(c)| \leq 1$, and a CNF is *Horn* if all its clauses are Horn. As well-known, a theory Σ is Horn iff it is represented by some Horn CNF. Moreover, all prime implicates of a Horn Σ are Horn.

A Horn theory Σ_c is a *Horn core* of a theory Σ [10,11], if $\Sigma_c \subseteq \Sigma$ and no Horn theory Σ' exists such that $\Sigma_c \subset \Sigma' \subseteq \Sigma$. Observe that, in general, Σ has more than one Horn core; e.g., $\Sigma = \{(110), (101)\}$ has two Horn cores $\Sigma_c^1 = \{(110)\}$ and $\Sigma_c^2 = \{(101)\}$, respectively. The *Horn envelope* of Σ [10,11] is the Horn theory $\Sigma_e \supseteq \Sigma$ such that no Horn theory Σ' such that $\Sigma_e \supset \Sigma' \supseteq \Sigma$ exists. For the above Σ, $\Sigma_e = \{(110), (101), (100)\}$. As easily seen, the Horn envelope is always unique. Let φ be a formula representing a theory Σ, and let φ_c and φ_e be formulas representing a Horn core and the Horn envelope of Σ, respectively. Then φ_c and φ_e are also called a *Horn core* and the *Horn envelope* of φ, respectively.

3 Formula-Based Representations

Our first result is that deciding the Horn property for a disjunction of Horn CNFs is intractable in general.

Theorem 1. *Given Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$, deciding if $\varphi = \bigvee_{i=1}^{l} \varphi_i$ is Horn is* co-NP*-complete.* □

The reduction in the hardness part of the proof of this theorem uses a number of Horn theories which depends on the instance of the reduction. For the case where l is bounded by some constant k, we have a positive result. We need further notation first.

Let $\varphi_i = \bigwedge_{j=1}^{m_i} c_{i,j}, \quad i = 1, 2, \ldots, l$. Denote by $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ the set of all nontautological Horn clauses c such that $N(c) = \bigcup_{i=1}^{l} N(c_{i,i_j})$ and $P(c) = P(c_{i,i_j})$ for some i, where $i_j \in \{1, 2, \ldots, m_i\}$ selects one clause for each i.

Example 1. Let $\varphi_1 = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_2 \vee \overline{x}_5 \vee x_6)$ and $\varphi_2 = (\overline{x}_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_3 \vee x_6)$. Then $HC(\varphi_1, \varphi_2) =\{(\overline{x}_1 \vee \overline{x}_3 \vee x_4), (\overline{x}_1 \vee \overline{x}_3 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)\}$. □

The following lemma states that any Horn theory represented by a disjunction of Horn CNFs has a CNF formula consisting of clauses only from $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$.

Lemma 1. *Let $\varphi_1, \varphi_2, \ldots, \varphi_l$ be Horn CNFs, and let $\varphi = \bigvee_{i=1}^{l} \varphi_i$. Then φ represents a Horn theory if and only if $\varphi \equiv \bigwedge_{c \in S} c$ for some subset $S \subseteq HC(\varphi_1, \varphi_2, \ldots . \varphi_l)$.*

Proof. The if-part is obvious because all clauses $c \in HC(\varphi_1, \varphi_2, \ldots . \varphi_l)$ are Horn. For the only-if-part, let φ represent a Horn Σ. Observe that φ is equivalent to the CNF

$$\varphi' = \bigwedge_{i_1=1}^{m_1} \bigwedge_{i_2=1}^{m_2} \cdots \bigwedge_{i_l=1}^{m_l} (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l}). \tag{3.1}$$

Since φ' also represents a Horn theory Σ and all prime implicates of a Horn theory are Horn, for each nontautological clause $c' = (c_{1,i_1} \vee c_{2,i_2} \vee \cdots \vee c_{l,i_l})$ in φ', there exists a Horn prime implicate c of Σ satisfying $\varphi' \leq c \leq c'$. Note that $HC(\varphi_1, \varphi_2, \ldots . \varphi_l)$ contains all maximal Horn clauses c^* with $c^* \leq c'$. Some clause c^* in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ satisfies $\varphi' (\equiv \varphi) \leq c \leq c^* \leq c'$. Thus, replacing each c' in φ' by such a c^* again produces Σ, which implies the only-if-part. □

Theorem 2. *For l bounded by a constant k, the problem in Theorem 1 can be solved in polynomial time.*

Proof. (Sketch) By Lemma 1, for each c' in φ' of (3.1), we only need to find a clause c^* in $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ such that $\varphi \leq c^* \leq c'$. If each c' has such a c^*, we can conclude that φ is Horn; otherwise, we can conclude that φ is not Horn. As for the time complexity, since $\varphi \leq c^*$ is equivalent to the condition that $\varphi_i \leq c^*$ holds for all i, and since each φ_i is Horn, checking $\varphi \leq c^*$ is polynomial. Furthermore, we have at most a polynomial number of c' and c^*. Thus, the overall time is polynomial. □

Let us now consider computing a Horn core. The following proposition, together with Theorem 1, implies that this is a difficult problem in general as well.

Proposition 1. *A theory Σ has a unique Horn core if and only if Σ is Horn.* □

Corollary 1. *Given Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$, computing a Horn core of $\varphi = \bigvee_{i=1}^{l} \varphi_i$ is* co-NP*-hard.* □

This is a rather negative result. The proof does not apply for a small (bounded by a constant) number of Horn theories, though. We next show that for two Horn theories ($l = 2$), the problem is polynomial.

3.1 Horn Cores of the Disjunction of Two Horn Theories

We start with the following lemma showing that any Horn core of a disjunction of Horn CNFs can be represented by a Horn CNF consisting of only clauses from $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$. By Proposition 1, this can be seen as a generalization of the only-if-part of Lemma 1. It follows from a similar result for CNF theories.

Lemma 2 (cf. [10]). *Let $\varphi_1, \varphi_2, \ldots, \varphi_l$ be Horn CNFs, and let ψ be any Horn core of $\varphi = \bigvee_{i=1}^{l} \varphi_i$. Then $\psi \equiv \bigwedge_{c \in S} c$ holds for some subset $S \subseteq HC(\varphi_1, \varphi_2, \ldots . \varphi_l)$.*

Proof. (Sketch) As φ is equivalent to the CNF φ' of (3.1) and ψ is a Horn core of φ, each nontautological clause c' in φ' is subsumed by some prime Horn implicate c of ψ (i.e., $c' \geq c$). Since $HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ is the set of all the maximal Horn clauses subsuming at least one clause in φ', some $c^* \in HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ exists such that $c \leq c^* \leq c'$. The CNF ψ' obtained from φ' by replacing each c' by such a c^* satisfies $\psi \leq \psi' \leq \varphi'$ ($\equiv \varphi$); this implies $\psi \equiv \psi'$. □

Note that the converse is not true in general, even if ψ is the CNF obtained from φ' as in (3.1) by replacing each nontautological clause c' in φ' by a clause $c^* \in HC(\varphi_1, \varphi_2, \ldots, \varphi_l)$ with $c^* \leq c'$. The following example gives such an instance.

Example 2. Let φ_1 and φ_2 be as in Example 1. Then φ' of (3.1) is written as $\varphi' = (\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)$ (where we exclude the tautological clauses from φ'). Let us consider a Horn CNF

$$\psi = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6).$$

Note that all clauses in ψ are from $HC(\varphi_1, \varphi_2)$ (see Example 1), and that ψ is obtained from φ' in the desired way, as $(\overline{x}_1 \vee \overline{x}_3 \vee x_4) \leq (\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6)$, $(\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5) \leq (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)$ and $(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6) \leq (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)$. However, ψ is not a Horn core of $\varphi = \varphi_1 \vee \varphi_2$, since

$$\psi' = (\overline{x}_1 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6) \qquad (3.2)$$

satisfies $\psi < \psi' \leq \varphi' \, (\equiv \varphi)$, which shows that ψ is not Horn core of φ. In fact, $\psi \leq \psi'$ follows from $\psi \leq (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)$, and this combined with $\psi(110111) = 0$ and $\psi'(110111) = 1$ implies $\psi < \psi'$. As shown later, ψ' of (3.2) is a Horn core of φ. □

Now we give an algorithm which computes a Horn core of the disjunction of two Horn theories.

Algorithm CORE
Input: Horn CNFs $\varphi_1 = \bigwedge_{i=1}^{m_1} c_{1,i}$ and $\varphi_2 = \bigwedge_{j=1}^{m_2} c_{2,j}$.
Output: A Horn core ψ of $\varphi = \varphi_1 \vee \varphi_2$.

```
Step 1. $S := \{c^*_{i,j} = c_{1,i} \cup c_{2,j} \mid c^*_{i,j} \not\equiv \top,\ i = 1, 2, \ldots, m_1,\ j = 1, 2, \ldots, m_2\}$;
        $S_2 := \{c \in S \mid |P(c)| = 2\}$.
            Let $c^1_{i,j}$ (resp., $c^2_{i,j}$) be the Horn clause $c$ such that
            $N(c) = N(c^*_{i,j})$ and $P(c) = P(c_{1,i})$ (resp., $P(c) = P(c_{2,j})$).
Step 2. $S_a := S \setminus S_2$; $S_b := S_2$;
        for each $c^*_{i,j} \in S_2$ do
          if $\varphi_1 \leq c^1_{i,j}$ and $\varphi_2 \leq c^1_{i,j}$ then
          begin $S_a := S_a \cup \{c^1_{i,j}\}$; $S_b := S_b \setminus \{c^*_{i,j}\}$ end
          elseif $\varphi_1 \leq c^2_{i,j}$ and $\varphi_2 \leq c^2_{i,j}$ then
          begin $S_a := S_a \cup \{c^2_{i,j}\}$; $S_b := S_b \setminus \{c^*_{i,j}\}$ end;
Step 3. Output $\psi := \bigwedge_{c \in S_a} c \wedge \bigwedge_{c^*_{i,j} \in S_b} c^1_{i,j}$.
```
□

Example 3. Let us apply algorithm CORE to Horn CNFs φ_1 and φ_2 given in Example 1. In Step 1, we have $S = \{(\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee \overline{x}_5 \vee x_6)\}$ and $S_2 = \{(\overline{x}_1 \vee \overline{x}_3 \vee x_4 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)\}$. The clause $c^*_{2,1} = (\overline{x}_1 \vee \overline{x}_2 \vee x_3 \vee \overline{x}_5 \vee x_6)$ satisfies the if-statement (i.e., $\varphi_1 \leq c^1_{2,1}$ and $\varphi_2 \leq c^1_{2,1}$ holds for $c^1_{2,1} = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_5 \vee x_6)$); the other clause in S_2 satisfies neither the if- nor the elseif-statement. Thus, Step 3 outputs the Horn CNF of (3.2). □

Observe that $HC(\varphi_1, \varphi_2)$ is the set of all clauses $c^1_{i,j}$ and $c^2_{i,j}$, and that this algorithm runs in polynomial time. Indeed, both implication tests in the "if" of Step 2 are solvable in linear time since φ_1 and φ_2 are Horn (cf. [5]), and all other steps are clearly polynomial.

Theorem 3. *Let φ_1 and φ_2 be Horn CNFs. Then, algorithm* CORE *computes a Horn core ψ of $\varphi = \varphi_1 \vee \varphi_2$ in polynomial time. Moreover, ψ includes φ_1, i.e., $\varphi_1 \leq \psi$ holds.*

Proof. (Sketch) Observe that

$$\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2 \tag{3.3}$$

obviously holds. Indeed, each clause c in S_a of Step 3 is an implicate of φ_1, and each clause $c^1_{i,j}$ is subsumed by some clause of φ_1; hence, the first implication holds. The second holds since $\varphi_1 \vee \varphi_2 \equiv \bigwedge_{c^*_{i,j} \in S} c^*_{ij}$ holds by definition, and each clause $c = c^*_{i,j}$ in $\varphi_1 \vee \varphi_2$ is subsumed by some clause in ψ.

It can be shown that ψ is one of the maximal Horn CNFs ψ^* such that $\varphi_1 \leq \psi^* \leq \varphi_1 \vee \varphi_2$ holds, i.e., it is a Horn core of $\varphi_1 \vee \varphi_2$. □

We mention that the algorithms for computing a Horn core in [10,2,1] are not polynomial on the disjunction of two Horn CNF formulas; they require a CNF for input.

An analysis of the algorithm CORE reveals that it bears no nondeterminism in computing a Horn core ψ satisfying $\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2$. This raises the suspicion that a Horn core including φ_1 might be unique. This is in fact the case.

Proposition 2. *Let Σ_1 and Σ_2 be Horn theories, and let $\Sigma = \Sigma_1 \cup \Sigma_2$. Then, there exists a unique Horn core Π that satisfies $\Sigma_1 \subseteq \Pi \subseteq \Sigma$.* □

A a consequence, we call the two Horn cores Π_1 and Π_2 satisfying $\Sigma_1 \subseteq \Pi_1 \subseteq \Sigma_1 \cup \Sigma_2$ and $\Sigma_2 \subseteq \Pi_2 \subseteq \Sigma_1 \cup \Sigma_2$, respectively, the *canonical Horn cores* of Π_1 and Π_2 with respect to $\Sigma = \Sigma_1 \cup \Sigma_2$. Observe that the generalization of Proposition 2 to a disjunction of $k(\geq 3)$ Horn theories does not hold. We note the following structural result.

Proposition 3. *Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_l$ be Horn theories. Then $\Delta = \bigcap_{i=1}^{l} \Sigma_i$ is contained in every Horn core of $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$.* □

3.2 Horn Cores of the Disjunction of more than Two Horn Theories

Let us first show that the direct generalization of algorithm CORE does not produce a Horn core for the case $l \geq 3$.

Example 4. Let $\varphi_0 = (\overline{x}_1 \vee \overline{x}_2 \vee x_4)(\overline{x}_2 \vee x_7)(\overline{x}_3 \vee x_5 \vee \overline{x}_7)$, $\varphi_1 = (\overline{x}_3 \vee x_5)$, and $\varphi_2 = (\overline{x}_1 \vee \overline{x}_3 \vee x_6)$, and let $\varphi = \varphi_0 \vee \varphi_1 \vee \varphi_2$. Then φ is equivalent to the CNF

$$\varphi' = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4 \vee x_5 \vee x_6)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5 \vee x_6 \vee x_7)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee x_6 \vee \overline{x}_7). \quad (3.4)$$

Since no c^* in $HC(\varphi_0, \varphi_1, \varphi_2)$ is an implicate of φ' $(\equiv \varphi)$ (note that, for $l = 2$, $HC(\varphi_1, \varphi_2)$ is the set of all clauses $c^1_{i,j}$ and $c^2_{i,j}$), S_b in Step 3 of the generalized algorithm CORE, applied to $\varphi_0, \varphi_1, \varphi_2$, is the set of all clauses appearing in φ' of (3.4). Thus, for the respective $S_a = \emptyset$ $(= \{c \in S \mid |P(c)| < 2\})$, we obtain

$$\psi = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_7)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

However, this ψ is not a Horn core of φ, since

$$\psi' = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7)$$

satisfies $\psi < \psi' \leq \varphi'$ $(\equiv \varphi)$. As shown in Example 5, ψ' is in fact a Horn core of φ. □

By using CORE repeatedly, we can give an algorithm to compute a Horn core of φ. This algorithm is polynomial if l is bounded by a constant k. Informally, it constructs a sequence of (not necessarily strictly) increasing Horn theories $\psi_0 \leq \psi_1 \leq \psi_2 \leq \cdots$ which is contained in φ. The sequence converges to ψ^*, which is a Horn core of φ.

Let $\varphi_0, \varphi_1, \ldots, \varphi_l$ be Horn CNFs, and let CORE(μ_1, μ_2) be a Horn CNF of the canonical Horn core of μ_1 with respect to $\mu_1 \vee \mu_2$. Define a sequence ψ_i, $i \geq 0$, as follows:

$$\begin{aligned}
\psi_0 &= \varphi_0;\\
\psi_1 &= \mathrm{CORE}(\psi_0, \varphi_1);\\
\psi_2 &= \mathrm{CORE}(\psi_1, \varphi_2);\\
\cdots &\quad \cdots\\
\psi_l &= \mathrm{CORE}(\psi_{l-1}, \varphi_l);\\
\cdots &\quad \cdots\\
\psi_{i\cdot l+j} &= \mathrm{CORE}(\psi_{i\cdot l+j-1}, \varphi_j), \qquad i \geq 0, \quad 1 \leq j \leq l\\
\cdots &\quad \cdots
\end{aligned}$$

This sequence ψ_i monotonically increases φ_0 to a Horn core of $\varphi = \bigvee_{i=0}^{l} \varphi_i$, by first increasing it in φ_1, then in φ_2 and so on. However, note that ψ_l is not necessarily a Horn core of φ. The reason is that in building ψ_1, say, some models of φ_1 may have been excluded which now can be added to the models of ψ_l such that the Horn property is preserved. To catch them, the algorithm loops and reconsiders φ_1, φ_2 etc. until no further change is possible.

Let us denote by

$$\psi^* \;=\; \lim_{i\to\infty} \psi_i$$

the limit of the sequence. Observe that $\psi^* = \psi_k$ holds for some finite $k \geq 0$.

Lemma 3. *For any Horn CNFs $\varphi_0, \varphi_1, \ldots, \varphi_l$, the above ψ^* is a Horn core of $\varphi = \varphi_0 \vee \varphi_1 \vee \ldots \vee \varphi_l$.*

Proof. (Sketch) By an inductive argument, it is easy to show that every ψ_i is a Horn CNFs satisfying $\psi_i \leq \varphi$. Hence, ψ^* is a Horn CNF satisfying $\psi^* \leq \varphi$. Assuming that ψ^* is not a Horn core of φ, we derive a contradiction.

In this case, as shown in [7], some model $v \in \Sigma \setminus \Pi^*$ exists such that $\Pi^* \cup \{v\}$ is Horn, where Π^* and Σ are the theories represented by ψ^* and φ, respectively. This v satisfies $v \in \Sigma_i$ for some i, where Σ_i is the Horn theory represented by φ_i. Let k be the index such that $\psi_k = \psi^*$. Then, since ψ^* is the limit, $\psi_{k'} = \psi^*$ holds for all $k' > k$. Consider the least $k' > k$ such that $\psi_{k'} = \mathrm{CORE}(\psi_{k'-1}, \varphi_i)$, i.e., the second argument is φ_i. Since $\psi_{k'-1} = \psi^*$ and $\psi_{k'-1}(v) = 0$ hold, ψ^* is not a Horn core of $\psi^* \vee \varphi_i$, a contradiction. □

Based on this lemma, we design the following algorithm CORE*.

```
Algorithm CORE*
Input: Horn CNFs $\varphi_i = \bigwedge_{j=1}^{m_i} c_{i,j}$, for $i = 0, 1, \ldots, l$.
Output: A Horn core $\psi$ of $\varphi = \varphi_0 \vee \varphi_1 \vee \cdots \vee \varphi_l$.

    Step 1. Set $\psi := \varphi_0$; $i := 0$; changes := 0;
    Step 2. while changes $< l$ do begin
                if $i < l$ then $i := i + 1$ else $i := 1$;
                $\psi_{new} :=$ CORE$(\psi, \varphi_i)$;
                if $\psi < \psi_{new}$ then
                begin  $S := \{c \in HC(\varphi_0, \varphi_1, \ldots, \varphi_l) \mid \psi_{new} \leq c\}$;
                    $\psi := \bigwedge_{c \in S} c$;  changes := 0;
                end
                else changes := changes + 1;
            end{while};
    Step 3. output a Horn CNF $\psi$.                                    □
```

Note that, during the iteration of Step 2, the size of a Horn CNF ψ_{new} might become large. Thus the algorithm replaces ψ_{new} by $\psi' = \bigwedge_{c \in HC(\varphi_0, \varphi_1, \ldots, \varphi_l): \psi_{new} \leq c} c$. By Lemma 2, this does not affect correctness. The algorithm repeatedly calls CORE with ψ as the first argument where the second argument loops within $\varphi_1, \varphi_2, \ldots, \varphi_l$. It halts, if no change within l subsequent calls is detected, i.e., CORE(ψ, φ_i) = CORE(ψ, φ_{i+1}) = ... = CORE(ψ, φ_{i-1}) holds, where $i \geq 1$ and $i - 1 = l$ if $i = 1$.

Example 5. Let us apply the algorithm CORE* to $\varphi = \varphi_0 \vee \varphi_1 \vee \varphi_2$, where φ_0, φ_1 and φ_2 are as in Example 4. In the first iteration of Step 2, using algorithm CORE from Subsection 3.1, we obtain

$$\psi_{new} = \text{CORE}(\varphi_0, \varphi_1) = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

We have $HC(\varphi_0, \varphi_1, \varphi_2) = \{ (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_6), (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_7), (\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7), (\overline{x}_1 \vee \overline{x}_3 \vee x_6 \vee \overline{x}_7) \}$; hence,

$$\psi_1 = (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_5)(\overline{x}_1 \vee \overline{x}_3 \vee x_5 \vee \overline{x}_7).$$

In the second iteration, we have $\psi_{new} = \text{CORE}(\psi_1, \varphi_2) = \psi_1$, and hence $\psi_2 = \psi_1$ holds. Since $l\,(= 2)$ consecutive members ψ_1 and ψ_2 satisfy $\psi_1 = \psi_2$, the algorithm outputs a Horn CNF ψ_2 and halts. □

Theorem 4. *Algorithm* CORE* *outputs a Horn core ψ of $\varphi = \varphi_0 \vee \varphi_1 \vee \cdots \vee \varphi_l$ satisfying $\varphi_0 \leq \psi \leq \varphi$. Moreover, it is polynomial if l is bounded by some constant.* □

4 Characteristic Models

For any Horn theory Σ, a model $v \in \Sigma$ is called *characteristic* [9], if $v \notin Cl_\wedge(\Sigma \setminus \{v\})$. The set of all characteristic models of Σ, the *characteristic set of* Σ, is denoted by $C^*(\Sigma)$. Note that every Horn theory Σ has the unique characteristic set $C^*(\Sigma)$. E.g., consider the Horn theory $\Sigma = \{(0101),$ $(1001),$ $(1000),$ $(0001),$ $(0000)\}$. Then $C^*(\Sigma) = \{(0101), (1001), (1000)\}$. In the rest of this section, we reconsider the problems in the previous section, assuming that Horn theories are represented by their characteristic models. We give here a summary of results and refer to [7] for further details.

Like in the case of CNFs, deciding whether a disjunction of Horn theories is Horn is intractable in general.

Theorem 5. *Given the characteristic sets $C^*(\Sigma_1)$, $C^*(\Sigma_2)$, ..., $C^*(\Sigma_l)$ of Horn Σ_1, Σ_2, ..., Σ_l, deciding whether $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ is Horn is* co-NP*-complete.* □

The next result gives a precise semantical characterization of the Horn property of a disjunction.

Theorem 6. *Let $M_i \subseteq \{0,1\}^n$, $i = 1, 2, \ldots, l$. Then $\Sigma = \bigcup_{i=1}^{l} Cl_\wedge(M_i)$ is Horn iff $\bigwedge_{v \in S} v \in \Sigma$ holds for every $S \subseteq \bigcup_{i=1}^{l} M_i$ such that $1 \leq |S| \leq l$.* □

As an immediate consequence, we obtain the following result.

Theorem 7. *For l bounded by a constant k, the problem in Theorem 5 is polynomial.* □

Theorems 5 and 7, respectively, coincide with Theorems 1 and 2, in which input Horn theories are represented by Horn CNFs.

For the case where Horn theories are represented by a CNF, we have presented in the previous section a polynomial-time algorithm for computing a Horn core ψ satisfying $\varphi_1 \leq \psi \leq \varphi_1 \vee \varphi_2$. For characteristic models, a similar algorithm is hard to be found. This is suggested by the following theorem, which tells that recognizing a Horn core is intractable, even for the case of two Horn theories.

Theorem 8. *Given the characteristic sets $C^*(\Sigma_1)$ and $C^*(\Sigma_2)$ of Horn theories $\Sigma_1, \Sigma_2 \subseteq \{0,1\}^n$, deciding whether Σ_1 is a Horn core of $\Sigma = \Sigma_1 \cup \Sigma_2$ is* co-NP*-complete.* □

We remark that computing the Horn envelope of a disjunction $\Sigma = \bigcup_{i=1}^{l} \Sigma_i$ is polynomial under characteristic model representation, while it requires in general exponential time (and space) if Horn CNFs are used. Indeed, the characteristic set of Σ is given by $C^*(M)$, where $M = \bigcup_{i=1}^{l} C^*(\Sigma_i)$. Clearly, $C^*(M)$ is computable from M in polynomial time, by repeatedly removing from M a model v which is represented by the intersection of other models in M, until no longer possible.

On the other hand, any Horn CNF φ representing Σ may be exponential in the size of Horn CNFs $\varphi_1, \varphi_2, \ldots, \varphi_l$ representing Σ_1,Σ_2,..., Σ_l, even if $l = 2$.

E.g., let $\varphi_1 = x_0$ and $\varphi_2 = (\overline{x}_1 \vee \overline{x}_2 \ldots \vee \overline{x}_m) \wedge \bigwedge_{j=1}^{m} (x_j \vee \overline{y}_j)$. Then $\varphi = \varphi_1 \vee \varphi_2$ is equivalent to the CNF $(x_0 \vee \overline{x}_1 \vee \overline{x}_2 \ldots \vee \overline{x}_m) \wedge \bigwedge_{j=1}^{m} (x_0 \vee x_j \vee \overline{y}_j)$. The Horn envelope of φ is represented by the conjunction of all Horn prime implicates of φ, i.e.,

$$\psi = \bigwedge_{z_1 \in \{x_1, y_1\}} \bigwedge_{z_2 \in \{x_2, y_2\}} \cdots \bigwedge_{z_m \in \{x_m, y_m\}} (x_0 \vee \bigvee_{j=1}^{m} \overline{z}_j).$$

As easily seen, no clause in ψ is redundant. Hence, ψ is the shortest Horn CNF for the Horn envelope of φ, and its size is exponential in the sizes of φ_1 and φ_2.

5 Conclusion

In this paper, we considered the Horn cores and the Horn envelope of a disjunction $\Sigma = \bigcup_i \Sigma_i$ of Horn theories Σ_i. We have proven that checking whether Σ is Horn is in general co-NP-complete for both representations, but is polynomially solvable if l is bounded by a constant. We have also shown that, if l is bounded by some constant, a Horn core Π of Σ satisfying $\Sigma_1 \subseteq \Pi \subseteq \Sigma$ can be computed from Horn CNFs in polynomial time, while it is co-NP-hard from characteristic models, even if $l = 2$. As for the Horn envelope, we have shown that it can be computed from characteristic models in polynomial time, but it cannot be computed from Horn CNFs, even if $l = 2$.

One of the further issues remaining for research is the problem of computing a Horn core Π of Σ (which might not satisfy $\Sigma_1 \subseteq \Pi \subseteq \Sigma$) from characteristic sets, in the case in which l is bounded by some constant.

Acknowledgments

We thank the referees for their helpful suggestions, and gratefully acknowledge the partial support of the Scientific Grant in Aid by the Ministry of Education, Science and Culture of Japan (Grant 06044112).

References

1. Y. Boufkhad. Algorithms for Propositional KB Approximation. In *Proc. National Conference on AI (AAAI '98)*, pp. 280–285, Madison, Wisconsin, July 26–30, 1998. 49, 55
2. M. Cadoli. Semantical and Computational Aspects of Horn Approximations. In *Proc. IJCAI-93*, pp. 39–44, 1993. 49, 55
3. M. Cadoli and M. Schaerf. Tractable Reasoning via Approximation. *Art. Intelligence*, 74(2):249–310, 1995. 49
4. A. del Val. An Analysis of Approximate Knowledge Compilation. In *Proc. IJCAI-95*, pp. 830–837, 1995. 49
5. W. Dowling and J. H. Gallier. Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Theories. *Journal of Logic Programming*, 3:267–284, 1984. 54
6. T. Eiter, T. Ibaraki and K. Makino. Computing Intersections of Horn Theories for Reasoning with Models. In *Proc. AAAI '98*, pp. 292–297, 1998. 49

7. T. Eiter, T. Ibaraki and K. Makino. Disjunctions of Horn Theories and their Cores. Manuscript, 1998. 50, 56, 58
8. G. Gogic, C. Papadimitriou, and M. Sideri. Incremental Recompilation of Knowledge. *J. Artificial Intelligence Research*, 8:23–37, 1998. 49
9. H. Kautz, M. Kearns, and B. Selman. Horn Approximations of Empirical Data. *Art. Intell.*, 74:129–245, 1995. 49, 50, 57
10. H. Kautz and B. Selman. Knowledge Compilation and Theory Approximation. *JACM*, 43(2):193–224, 1996. 49, 51, 53, 55
11. D. Kavvadias, C. Papadimitriou, and M. Sideri. On Horn Envelopes and Hypergraph Transversals. In W. Ng, ed., ISAAC-93, LNCS 762, 399–405, 1993. 49, 51
12. R. Khardon and D. Roth. Reasoning with Models. *Art. Intell.*, 87(1/2):187–213, 1996. 50
13. D. Roth. On the Hardness of Approximate Reasoning. *Art. Intell.*, 82(1/2):273–302, 1996. 49

Checking Programs Discreetly: Demonstrating Result-Correctness Efficiently While Concealing It

Giovanni Di Crescenzo[1], Kouichi Sakurai[2] *, and Moti Yung[3]

[1] Computer Science and Engineering Department, University of California
San Diego, La Jolla, CA, 92093-0114, USA
giovanni@cs.ucsd.edu
[2] Dept. of Computer Science and Comm. Eng., Kyushu Univ.
Fukuoka 812-81, Japan
sakurai@csce.kyushu-u.ac.jp
[3] CertCo, New York, NY, USA
moti@certco.com
moti@cs.columbia.edu

Abstract. We formalize and investigate a model for zero-knowledge proofs of "program result-correctness", which naturally extends Blum's theory of program checking by adding zero-knowledge requirements. The zero-knowledge requirements are universal for yes and no instances alike.

1 Introduction

Suppose a company has developed an efficient heuristic or an excellent algorithm to solve a problem, for which no efficient algorithm has yet been found (examples of such problems are "quadratic residuosity" (QR) [20] and "graph isomorphism" (GI) [26]). To sell or license the program, the company has to announce the availability of the new algorithm and its salesmen must *demonstrate* its correctness effectively. A serious problem for the salesman is that the demonstration should be carefully crafted: It should not reveal information to tricky customers who may seek getting important algorithmic details. Furthermore, while demonstrating on these tricky clients' instances, the actual algorithm's output should not be revealed (for free). We note that (1) the salesman need not know the algorithm itself (it should suffice that he knows how to demonstrate the correctness of the program by interacting with it), and (2) of course, during the demonstration he needs to use only the program and not, say, an additional communication to a super-computer.

1.1 Our Results and Relationships to Known Results

A starting approach to this problem may be to try to employ the theory of program checking introduced by Blum and Kannan in [12]. It has strong connections

* Partially done while visiting Columbia University.

with interactive proof systems of membership, introduced by Goldwasser Micali and Rackoff [22]. In fact, [12] concretely proposed a method for checking the correctness of a program deciding whether two given graphs are isomorphic or not by using an interactive proof system for membership of GI and one of its complement GNI of [21]. However, the salesman cannot use directly this protocol since it reveals the bit indicating isomorphic/non-isomorphic and eventually (via self reducibility) the isomorphism between the two input graphs as well. A second approach to the salesman's problem might be to try to directly use the theory of zero-knowledge interactive proof systems of membership. E.g., [22] proposed a protocol to show that a given integer is a quadratic residue (QR), and a different protocol to show that a given integer is a quadratic non-residue (QNR). Both protocols are zero knowledge and do not give additional information. If we try the combined protocol where if the integer is a quadratic residue, then the prover executes the first protocol and otherwise, the second. Unfortunately, this protocol still leaks this one-bit information. Also, it seems that we need to formulate the fact that the program really "knows" the output (rather than merely a proof of membership). Thus more care is needed.

The study of zero-knowledge proofs of knowledge which do not even reveal the one-bit information of whether a particular statement, is true or false was initiated by Feige, Fiat, and Shamir [16]; however, their model deals with a prover who has a special NP-knowledge. Recently Di Crescenzo, Sakurai, and Yung [14] gave new zero-knowledge protocol techniques and showed a perfect zero-knowledge protocol for proving the result-correctness of a program for a random self-reducible problem. A zero-knowledge proof of program result-correctness is an interactive protocol between an efficient prover and an efficient verifier. The goal is to transfer the correctness of the output returned by a program on a given input x and proving it correct without revealing any information. This model of [14] is a proper generalization of that of [16] (with no restriction to NP $\cap$ co-NP), motivated us to consider the possibility of zero-knowledge requirements in program checking [12]. In the context of [12] (as well as in the model of [8] which is a proof of membership motivated by program checking), limiting the power of the prover to the power of the program checked is important. In this paper we explore which programs can be demonstrated correct in a zero-knowledge fashion. Next we show why the models in [16,14] fall short of capturing this last constraint.

Perfect Zero-knowledge and subtlety with previous protocol techniques
We first consider methods to demonstrate result-correctness of program in perfect zero-knowledge, i.e., without revealing any information to infinitely-powerful adversaries. We observe that the protocol in [16] implies that a polynomial-time prover can show that he can decide whether a given input belongs to the language of quadratic residuosity modulo Blum integers or not. However, the prover requires the knowledge of the **complete factorization** of the modulus, which is believed to require greater computational power than only to **decide quadratic residuosity**. Thus, the discussion above fails to meet our original goal, in which

the prover requires only the minimal power of a correct program to solve QR and the prover is otherwise limited to efficient computation (thus, may not be able to factor numbers). A similar problem arises when we apply the general protocol in [14] for random self-reducible languages to the case of QR problem. Thus, no previous model and general technique induces the possibility of proving in perfect zero-knowledge the result-correctness of a program to solve QR. By designing a new protocol, this paper affirmatively answers to this problem for QR under the promise that the modulus is a Blum integer. We also notice that the protocol for GI in [14] works within the model formulated here as well.

Upper bounding the class of problems with perfect zk-protocols
We next give a theoretical upper bound to the class of problems having perfect/statistical zero-knowledge proofs of result-correctness, which suggests that such stronger security limits the power of these demonstration systems. A tool used by our proof is the upper bound to the power of the prover in statistical zero-knowledge proofs of membership given in [11], and our argument makes a new use of the (knowledge-)extractor.

The computational-zk case
Finally, we consider methods to demonstrate result-correctness of program in computational zero-knowledge, i.e., without revealing any information to a polynomially-bounded adversary. Feige et al. [16] showed, under the assumption that secure encryption schemes exist, the prover can show that it possesses a witness to a statement in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$ without telling anything new about it (not even whether the prover found a proof or a counterexample). We extend this result to PSPACE-complete decision problems under our model, and show that assuming one-way functions, the correctness of the program to solve PSPACE-complete decision problems can be demonstrated in computational zero-knowledge.

1.2 Related Work

The notion of "hiding information of inputs from an oracle" of Abadi, Feigenbaum and Kilian [1] considers the issue of the requester hiding its input from the program (or computational device), and its zero-knowledge aspect is investigated by Beaver, Feigenbaum and Shoup in [5]. Our model can be viewed as treating the opposite direction: how to protect the (results of) the program/computing-device.

Frankel, Gemmell, and Yung [17] introduced the notion of cryptographic program checking. They considered the security needs of the program owners as our work, thus we next discuss the difference between [17] and ours. In [17] model, after the interaction between the owner and the checker, the checker knows the (correctness) result $y = P(\alpha)$ of the program P of the owner for a given input α. Namely, the [17]'s requirement of security against the checker is leaking "no **additional** knowledge" as in [19,15] where the result is given to the verifier. Whereas our requirement of security is stronger: the checker **cannot learn anything**, even the result of the program P for a given input α as the security requirement in [16,14].

The approach developed in [17] is the "witness-based" approach. Namely, to use a (small) trusted collection of instances of inputs and their corrected outputs that the checker holds in advance. The idea of zero-knowledge checking by viewing the result for a randomly selected set of witnesses was given in Yung's "proof of computational power" [30] (which is another motivating model); but there the verifier should be able to sample these instances. In contrast, our model is more general: the checker assumes to hold no witnesses (just black box access to the output of a decision-problem solver, and no witness sampling). We remark that the "witness-based" approach [17] is working well for the problem with algebraic properties (such as random self-reducibility) [1]. In particular, it is useful for the application of robust *homomorphic* function sharing [13]. On the other hand, our model can be applied to the problem without such homomorphic properties (e.g. Graph Isomorphism). From complexity-theoretical point of view, the model in [17] can be regarded as a special case of the previous interactive proofs [22,19,12]. Thus, the known complexity results on the previous [22,19] can be applicable to model of [17]. However, our model is completely new, so no previous complexity theoretic result can directly apply. For example, showing a theoretical upper bound on the class proved in our model requires a new argument (e.g. for Theorem 3).

2 The Proofs of Result-Correctness: Model and Complexity

2.1 Notations

If A and B are two interactive probabilistic Turing machine, by pair (A,B) we denote an interactive protocol ([22]). Let x be an input common to A and B. By $\mathrm{tr}_{(\mathrm{A},\mathrm{B})}(x)$ we denote the transcript of an execution of protocol (A,B) on input x, that is, the messages written on V's communication tape during an execution of protocol (A,B) on input x. By $\mathrm{OUT}_{\mathrm{B}}(\mathrm{tr}_{(\mathrm{A},\mathrm{B})}(x))$ we denote B's output given the transcript $\mathrm{tr}_{(\mathrm{A},\mathrm{B})}(x)$.

If L is a language, by $\chi_L : \{0,1\}^* \to \{0,1\}$ we denote the characteristic function for the language L (i.e., $\chi_L(x) = 1$ if and only if $x \in L$).

2.2 Program Checking and Competitive Interactive Proofs

Before defining our new model, we recall the notions of program checking and competitive interactive proof systems. Program checkers was introduced in [12].

Definition 1 ([12]). Let C be a PPT oracle Turing machine. We say that C a checker for a function $F : \{0,1\}^* \to \{0,1\}^*$ if for all program π and all $x \in \{0,1\}^*$ it is the case that

1. If $\pi(y) = F(y)$ for all $y \in \{0,1\}^*$, then $\mathrm{Prob}(\mathrm{OUT}_C(\mathrm{tr}_{(\pi,\mathrm{C})}(x)) = \mathrm{ACCEPT}) \geq 1 - 2^{-|x|}$.

2. If $\pi(x) \neq F(x)$, then $\text{Prob}(\text{OUT}_C(\text{tr}_{(\pi,\text{C})}(x)) = \text{ACCEPT}) \leq 2^{-|x|}$.

Note that the probabilities are over the random coin tosses of the checker C.

The following type of interactive proof systems was considered in [8].

Definition 2 ([8]). Let A and B be two interactive probabilistic polynomial-time (PPT) Turing machines. We say that (A, B) is a *competitive* interactive proof for membership of a language L if the following two conditions hold.

1. For all $x \in L$, $\text{Prob}(\text{OUT}_B(\text{tr}_{(\text{A}^\text{L},\text{B})}(x)) = \text{ACCEPT}) \geq 1 - 2^{-|x|}$.
2. For all $x \notin L$, and all functions A^*, $\text{Prob}(\text{OUT}_B(\text{tr}_{(\text{A}^*,\text{B})}(x)) = \text{ACCEPT}) \leq 2^{-|x|}$.

2.3 Definition and Properties of Proof Systems of Result-Correctness

Informally, a proof system of program result-correctness is an interactive proof system between a PPT prover having access to the program to be proved correct and a PTT verifier. The prover's goal is to transfer the correctness of the output of the program on a given input x through this interactive protocol.

The formal definition for proofs of program result-correctness has two requirements: competitive regularity and validity, which we first informally describe. The regularity requirement states that the verifier accepts with high probability for any input x. The competitive regularity further requires that the PPT prover can make the verifier accept by just using the program as an oracle [8]. The validity requirement states that there exists an extractor that, for any input x, and interacting with any prover that forces the verifier to accept with 'sufficiently high' probability, is able to compute the correct output of the program (i.e., to compute the decision predicate that the program is supposed to compute), within a 'properly bounded' expected time. We remark that this differs from previous work on proofs of knowledge in which the extractor existed only for input in the language and was required to output a string satisfying a polynomial relation with the input. Our approach allows to consider even languages beyond NP. The basic approach in defining the validity requirement follows the definition for proofs of knowledge given in [7].

Definition 3. Let $P : \{0,1\}^* \to \{0,1\}$ be a decision problem, $err : \{0,1\}^* \to [0,1]$ be a function, and let π_P be a program supposed to compute P. Also, let A and B be two interactive PPT Turing, where A has access to program π_P. We say that (A,B) is a *proof system of program result-correctness* with knowledge error err for P if:

1. (Competitive Regularity)
   If for all x, $\pi_P(x) = P(x)$ then $\text{Prob}(\text{OUT}_B(\text{tr}_{(\text{A},\text{B})}(x)) = \text{ACCEPT}) \geq 1 - 2^{-|x|}$.
2. (Validity) There exists a probabilistic oracle machine E such that for all x and any Turing machine A′, and letting $acc_{A'}(x) = \text{Prob}(\text{OUT}_B(\text{tr}_{(\text{A}',\text{B})}(x)) = \text{ACCEPT})$, if $acc_{A'}(x) > err(x)$ then,

– $\mathrm{Prob}(\mathrm{OUT}_E(\mathrm{tr}_{(\mathrm{A}',\mathrm{E})}(x)) = P(x)) \geq 1 - 2^{-|x|}$.
– The machine E halts within expected time bounded by $\frac{|x|^c}{(acc_{A'}(x) - err(x))}$, for some constant $c > 0$.

2.4 Complexity of Proof-Systems of Result-Correctness

If the decision problem P has a proof system of program result-correctness, we will also say that the language L has a proof system of program result-correctness, where L is the set of strings x such that $P(x) = 1$. Clearly, any competitive interactive proof of membership for a language L is a proof of program result-correctness for the language L. In particular, from [29], we obtain that there is a proof system of program result-correctness for PSPACE. We observe that the above definition can be easily generalized to search problems. Here, using the result of [27] one obtains that any $\#P$-complete function has a proof system of program correctness. Furthermore, we can give the following upper bound for the model of proofs of program result-correctness.

Proposition 1 *If there exists a result-correctness proof system (with knowledge error 0) for a program π deciding a language L, then L is in PSPACE.*

To show that this upper bound is true, one can just observe that the honest prover can be clearly implemented in PSPACE and the extractor runs in expected polynomial-time when interacts with the honest prover; then just run an algorithm in $ZPP^{PSPACE} = PSPACE$ that simulates the interaction between the prover and the extractor.

Babai, Fortnow, and Lund [4] proved that every EXP-complete language has a function-restricted interactive proof system (namely, a proof system where the honest prover decides the language and the dishonest prover is a function from the set of instances to { yes,no }). This, using the checker characterization theorem of [12], implies that every EXP-complete language has a program checker. Thus, we conclude that:

Proposition 2 *Not all languages having a program checker have a proof system of result-correctness (with knowledge error 0) unless EXP = PSPACE.*

2.5 Security against Verifiers

We now add security concerns to the newly introduced proof-systems. The zero-knowledge requirement for these proof systems of program result-correctness is defined similarly as for interactive proof systems of membership. We first recall that the *view* of the verifier is everything he reads from his tapes during an interaction with the prover: formally we define $View_{\mathrm{V}}(x)$, V's view of the interaction with P on input x, as the probability space that assigns to pairs $(R; \mathrm{tr}_{\mathrm{P,V(R)}}(x))$ the probability that R is the tape of V's random bits and that $\mathrm{tr}_{\mathrm{P,V(R)}}(x)$ is the transcript of a conversation between P and V on input x given that R is V's random tape.

We notice that in the case of proofs of program result-correctness for a language L, the zero-knowledge condition requires that the simulator exists both if $x \in L$ and if $x \notin L$ (i.e. for any input $x \in \{0,1\}^*$), whereas a zero-knowledge proof for membership for a language L requires the existence of the simulator only when $x \in L$.

Definition 4. Let (A,B) be an interactive proof system of program result-correctness with knowledge error *err* for P. We say that (A,B) is a *perfect zero-knowledge proof system of program result-correctness* with knowledge error *err* for P if for each B′ there exists a probabilistic machine $S_{\mathrm{B}'}$ running in expected polynomial time such that for all x the probability spaces $\mathit{View}_{\mathrm{B}'}(x)$ and $S_{\mathrm{B}'}(x)$ are equal.

Intuitively, the definition above says that the verifier cannot get any information after its interaction with the prover, not even the value of $P(x)$.

We note that in our previously developed protocol for GI [14], the prover is competitive (since GI-search reduces to GI-decision), which implies:

Theorem 1. Graph Isomorphism has a perfect zero-knowledge proof of program result-correctness (with knowledge error 0).

We also consider more restricted security according to the level (namely, statistical and computational) of indistinguishability.

Definition 5. Let (A,B) be an interactive proof system of program result-correctness with knowledge error *err* for P. We say that (A,B) is a *statistical zero-knowledge proof system of program result-correctness* with with knowledge error *err* for P if for each B′ there exists a probabilistic machine $S_{\mathrm{B}'}$ (called the simulator) running in expected polynomial time such that for all x, the probability spaces $\mathit{View}_{\mathrm{B}'}(x)$ and $S_{\mathrm{B}'}(x)$ are statistically indistinguishable; that is, for any constant c and any x, $|\sum_\alpha (\mathit{prob}(\mathit{View}_{\mathrm{B}'}(x) = \alpha) - \mathit{prob}(S_{\mathrm{B}'}(x) = \alpha))| < 1/|x|^c$.

Definition 6. Let (A,B) be an interactive proof system of program result-correctness with knowledge error *err* for problem P. We say that (A,B) is a *computational zero-knowledge proof system of program result-correctness* with knowledge error *err* for P if for each B′ there exists a probabilistic machine $S_{\mathrm{B}'}$ (called the simulator) running in expected polynomial time such that for all x, the probability spaces $\mathit{View}_{\mathrm{B}'}(x)$ and $S_{\mathrm{B}'}(x)$ are computationally indistinguishable; that is, for any poly-size family of circuits $\{C_n\}$, for any constant c and any x, $|\mathit{prob}(C_n(\mathit{View}_{\mathrm{B}'}(x)) = 1) - \mathit{prob}(C_n(S_{\mathrm{B}'}(x)) = 1)| < 1/|x|^c$.

3 A Perfect Zero-Knowledge Protocol for QR Modulo Blum Integers

We consider the problem of proof system of program result-correctness which is perfect zero-knowledge. Our result is a protocol for the language of quadratic residuosity in the case the modulus is a Blum integer [10].

3.1 Blum Integers and Related Properties

We informally state some properties about Blum integers and quadratic residuosity, see [28,20] for formal descriptions and proofs.

An integer x is a Blum integer if and only if $x = p^{k_1}q^{k_2}$, where p and q are different primes satisfying $p \equiv q \equiv 3 \bmod 4$ and k_1 and k_2 are odd integers. Let x be a Blum integer, and let Z_x^* be the multiplicative group modulo x. Define Z_x^{+1} and Z_x^{-1} to be, respectively, the sets of elements of Z_x^* with Jacobi symbol $+1$ and -1. Define the set $QR_x = \{y \in Z_x^* \,|\, \exists u \in Z_x^* \,:\, y = u^2 \bmod x\}$ of quadratic residues modulo x, and the set $QNR_x = \{y \in Z_x^{+1} \,|\, \not\exists u \in Z_x^* \,:\, y = u^2 \bmod x\}$ of quadratic non residues modulo x. If x is a Blum integer, then $-1 \bmod x$ is a quadratic non residue with Jacobi symbol $+1$. This implies that on input a Blum integer x, it is easy to generate a random quadratic non residue in Z_x^{+1}: randomly select $r \in Z_x^*$ and output $-r^2 \bmod x$. Moreover, for this special class of integers we have that for any $y_1, y_2 \in Z_x^*$, the product y_1y_2 is a quadratic residue if and only if both y_1, y_2 are residues or both are non-residues. The quadratic residuosity assumption states that it is hard to distinguish the two sets QR_x, QNR_x and is widely used in the construction of systems for various cryptographic applications. The better algorithm known to distinguish QR_x from QNR_x requires the factorization of the modulus, another well-known problem which is believed hard.

3.2 A Competitive Protocol for $\mathbf{QR}_x$

Competitive interactive proofs of membership for quadratic residuosity have been investigated in [8,24]. Our protocol is different in two aspects. Namely, it is a proof of program result-correctness and it is perfect zero-knowledge: thus, it allows the prover to show that his program returns the correct output on a common input y, without revealing whether $y \in QR_x$ or $y \in QNR_x$, or any other additional information.

First, we recall two simple observations. As already noticed in [8], the proof system of membership for QNR_x given in [22], where x may be any integer, is competitive. Then, if x is a Blum integer, one obtains a competitive proof system of membership for QR_x as follows: on input v, the prover and the verifier run the competitive protocol for QNR_x on input $-v \bmod x$. Both competitive proof of membership will be used as subprotocols in our proof of program result-correctness for QR_x. Then, we can informally describe our protocol (A,B) in the following way: on input y, A uniformly generates two integers z_0, z_1 such that exactly one has the same quadratic residuosity as the input y. Then B sends two random bits b, c. Now, if $b = 0$, A reveals the random coins used to compute z_0, z_1. In this step A shows that z_0, z_1 have been correctly computed. On the other hand, if $b = 1$ then A considers the integer z_c sent before: if z_c is a quadratic residue, then he interactively proves the statement '$z_c \in QR_x$' to B, using the above competitive proof of membership; if z_c is a quadratic non residue, then he interactively proves the statement '$z_c \in QNR_x$' to B, using the above competitive proof of membership. In this step A shows that if z_0, z_1 have been correctly computed, then he has a program that can decide whether y is a quadratic residue or not. B accepts if he is convinced from such proof. We denote by (C,D) the proof system of membership given in [22] for quadratic non-residuosity. A formal description of (A,B) is as follows.

The Protocol (A,B)

A: Uniformly choose a bit a and two numbers $r_0, r_1 \in Z_x^*$; set $z_a = y \times (r_a)^2 \bmod x$ and $z_{1-a} = -y \times (r_{1-a})^2 \bmod x$. Send (z_0, z_1) to B.

B: Uniformly choose two bits b, c, and send (b, c) to A.

A: Send a to B. If $b = 0$ then send r_0, r_1 to B. Otherwise if $z_c \in QR_x$, set $d = 1$ else set $d = 0$; send d to B and prove that $(-1)^d \times z_c \bmod x \in QNR_x$ (this step is executed as follows: A and B run the protocol (C,D) on input $(-1)^d \times z_c \bmod x$; where A runs C's program and B runs D's program).

B: If $b = 0$ check that z_0, z_1 are correctly constructed; if $b = 1$ check that the proof that $(-1)^d \times z_c \bmod x \in QNR_x$ is convincing.

It is possible to prove that (A,B) is a perfect zero-knowledge proof of program result-correctness with error 1/2 for QR_x. Define the protocol (P,V) as the sequential execution of n copies of (A,B), where V accepts if B does not reject in any of the n iterations. Thus, we obtain the following

Theorem 2. The protocol (P,V) is a perfect zero-knowledge proof of program result-correctness (with knowledge error 0) for the language QR_x

Proof. Competitive regularity. The requirement of competitive regularity follows from the properties of the two subprotocols, which are also competitive, as seen before.

Validity. To see that the validity requirement is satisfied, observe that we can construct the following extractor E (we omit here the analysis of the trivial cheating strategies by P′). On input y, E runs in parallel the two following algorithms, and halts when any of the two outputs first. The first algorithm is an exponential-time search algorithm to compute whether the input y is a quadratic residue modulo x or not (such an algorithm may be for instance to enumerate all integers $z \leq x$ and see if $z \in Z_x^*$ and $z^2 \bmod x$; y is a quadratic residue if and only if such a z is found; and the running time of this algorithm is at most $|x|^k \cdot 2^{|x|}$, for some constant k). The second algorithm is the following: E runs the program of the verifier V, interacting with P′, and then completely rewinds the prover P′ and starts again, using independently and uniformly chosen random bits, until he gets two accepting conversations such that $b_j^1 \neq b_j^2$ for some $j = 1, \ldots, n$, where by b_j^i we denote bit b sent by E in the j-th sequential repetition of the i-th accepting conversation. If he has obtained these two accepting conversations, he has obtained also a pair of integers (r_a, r_{1-a}) such that $r_a^2 \times y = z_a \bmod x$, $-r_{1-a}^2 \times y = z_{1-a} \bmod x$ and a convincing proof that one of z_a, z_{1-a} is a quadratic residue or non-residue modulo x. This clearly allows him to obtain the quadratic residuosity of y (for instance, if z_{1-a} is proved as a quadratic residue modulo x, then y is a quadratic residue).

To show that the result is correct with high probability, we observe that if E runs the exponential-time search algorithm, then clearly his output is correct with probability 1. Otherwise, the probability that E returns a wrong value for the quadratic residuosity of y is at most the probability that a cheating P′ is able to convince him that z_{c_j} is quadratic residue while it is not (or vice versa), and this probability is negligible from the soundness of the proof system of membership used.

To show that the expected running time is properly bounded (i.e., polynomially proportional to the reciprocal of the acceptance probability of prover P′), we observe the following. We consider two cases according to whether the number of accepting conversations that may be possibly obtained interacting with P′ is 1 or at least 2 (if such number is 0, we don't need to prove anything, since $acc_{P'}(x) = 0$). Assume such number is 1. Then E will output because of the exponential-time algorithm since he will never obtain two accepting conversations from P′. However, $acc_{P'}(x) \leq 2^{-|x|}$, since there exist exactly one value of random bits $b_1, \ldots, b_n$, for which the conversation is accepting, and thus the expected running time is bounded by $|x|^k \cdot 2^{|x|} = |x|^k / acc_{P'}(x)$. Now, assume the number of accepting conversations that may be possibly obtained interacting with P′ is at least 2. Then E may output because of the fact that he obtained 2 accepting conversations from P′. To see that E's time is properly bounded even in this case, observe that the expected number of rewindings of P′ needed to obtain two accepting conversation can be bounded as at most $3/acc_{P'}$. Then we just observe that at each rewinding E runs in polynomial time.

Perfect zero-knowledge. Consider a simulator S that uses at each iteration the standard trial and error strategy. Namely, S will prepare a pair (z_0, z_1) such that he can answer either to $b = 0$ or to $b = 1$, randomly choosing the case; then, he guesses V′ question b with probability 1/2, and thus output in expected polynomial-time. To prepare the pair (z_0, z_1) so that he can answer to case $b = 0$, he randomly chooses $a \in \{0, 1\}$ and $r_0, r_1 \in Z_x^*$ and then computes $z_a = y \times (r_a)^2 \bmod x$ and $z_{1-a} = -y \times (r_{1-a})^2 \bmod x$. The answers will be r_0, r_1. To prepare the pair (z_0, z_1) so that he can answer to case $b = 1$, he randomly chooses $a \in \{0, 1\}$ and $r_0, r_1 \in Z_x^*$ and then computes $z_a = (r_a)^2 \bmod x$ and $z_{1-a} = -(r_{1-a})^2 \bmod x$. In this case S knows the quadratic residuosity of z_c and thus can simulate the proof by P by running the simulator for the protocol for QNR_x on input z_c or $-z_c \bmod x$ according to which is the case. Observe that the output of $S_{V'}$ and the execution of protocol (P,V′) on input y are equally distributed. In particular, we observe that the pair (z_0, z_1) is equally distributed as a randomly ordered pair of a random quadratic residue and a random quadratic non-residue for both values of b.

4 An Upper Bound for Statistical ZK Proofs of Result-Correctness

We give an upper bound on the class of languages having perfect/statistical zero-knowledge proof systems of result-correctness. Namely, we prove the following

Theorem 3. Let L be a language. If L has a statistical zero-knowledge proof system of program result-correctness (with knowledge error 0), then L is in BPP^{NP}.

Basic idea and techniques used: We use a result by Bellare and Petrank [11] that investigated how much power is sufficient for the prover to execute a statistical zero-knowledge protocol. Namely, they showed that any statistical zero-knowledge proof system of membership for a language L can be modified into a statistical zero-knowledge proof system for L, in which the prover runs in PPT with an NP-oracle. The same construction works also for any statistical zero-knowledge interactive protocol and thus also for proof systems of result-correctness.

Then a sketch of our construction is the following: given a statistical zero-knowledge proof system of result-correctness (A,B) with knowledge error 0 for L, we apply the mentioned transformation to A and obtain a statistical zero-knowledge proof system of result-correctness (M^{NP},B) with negligible knowledge error. Then (M^{NP},B) has an extractor E that can decide L within a 'properly bounded' expected time (specifically, polynomially and inversely proportional to the acceptance probability of the possibly cheating prover P' that he is interacting with). Then one can construct a BPP^{NP} algorithm D which simulates a polynomial number of executions of E and runs the machine M^{NP} to answer to E's queries. Clearly, at least one of E's executions run by D will halt; and, finally D returns E's output. Details of construction and proof are omitted.

5 Computational Zero-Knowledge Proofs of Program Result-Correctness

This section studies the power of computational zero-knowledge proofs of program result-correctness and prove that all PSPACE-complete languages have such a proof under the assumption of existence of secure probabilistic encryption schemes ([20]). A tool we use is the following theorem given in [25,9].

Theorem 4. [25,9] Let (A,B) be an interactive proof system of membership for a language L. If secure bit commitment schemes exist, then there is a computational zero-knowledge proof system of membership (P,V) for L.

We show that:

Theorem 5. Let L be any PSPACE-complete language. If secure bit commitment schemes exist, then (P,V) is a computational zero-knowledge proof system of program result-correctness (with knowledge error 0) for L.

We informally describe our construction. Since PSPACE is closed under complement, and using Shamir's result [29], we can construct an interactive proof system (A_1,B_1) of membership to L and an interactive proof system (A_0,B_0) of membership to $\overline{L}$. Now, both proof systems (A_1,B_1) and (A_0,B_0) are transformed into Arthur-Merlin proof systems with perfect completeness. A further step is that of padding the messages exchanged in the two above proof systems in some standard way, so that the number of rounds, the length of the messages sent by the provers and of those sent by the verifiers, are equal in both protocols. Let (A'_1,B'_1) and (A'_0,B'_0) be the two protocols resulting from such transformations. Notice that B'_1 and B'_0 now differ only in the accepting predicate in the end. Then, consider the following proof system of membership (A,B) for the language $\{0,1\}^*$. On input x, A computes $\chi_L(x)$ and sends a bit $b = \chi_L(x)$ to B. Then A and B run protocol (A'_b,B'_b). B accepts if B'_b accepts the conversation with A'_b. The proof of this protocol's properties is omitted.

Open Problems Finding perfect/statistical zero-knowledge proofs of program result-correctness for other problems is interesting. A stronger upper bound on the class of languages having statistical zero-knowledge proof systems, namely that $\mathsf{SZKIP} \subseteq \text{AM} \cap \text{co-AM}$, has been given in [18,2]. The proof technique they use does not extend to proofs of program result-correctness. Whether this bound can be obtained also for statistical zero-knowledge proofs of program result-correctness is interesting as well.

Acknowledgments

Thanks to Russell Impagliazzo for useful discussions, in particular, on the upper bound of proofs of program result-correctness (in Proposition 1).

References

1. M.Abadi, J.Feigenbaum, and J.Kilian, *On Hiding Information from an oracle,* in J. Comput. System Sci. Vol.39, No.1, 1989, pp. 21–50. 61, 62
2. W. Aiello and J. Håstad, *Statistical Zero Knowledge Can Be Recognized in Two Rounds,* in J. Comput. System Sci. 42, 1991, pp. 327–345. 70
3. L. Babai, and L.Fortnow, "A characterization of #P by arithmetic straight line programs," in FOCS'90.
4. L.Babai, L.Fortnow, and C.Lund, "Non-deterministic exponential time has two-prover interactive protocols," in FOCS '90. 64
5. D.Beaver, J.Feigenbaum, and V.Shoup, "Hiding instances in zero-knowledge proof systems," in Proc. of CRYPTO'90. 61
6. L. Babai and S. Moran, *Arthur–Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes*, in JCSS v. 36, 1988, pp. 254–276.
7. M. Bellare and O. Goldreich, *On Defining Proofs of Knowledge*, in Proc. of CRYPTO '92. 63
8. M. Bellare and S. Goldwasser, *Languages that are Easier than their Proofs*, in SIAM J. on Computing. 60, 63, 66

9. M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, *Everything Provable is Provable in Zero Knowledge*, in Proc. of CRYPTO 88. 69
10. M. Blum, "Coin flipping by phone," in *IEEE Spring COMPCOM,* pp.133-137 (Feb.,1982). 65
11. M. Bellare and E. Petrank. *Making Zero-Knowledge Provers Efficient,* in STOC 92, 1992, pp. 711–722. 61, 69
12. M. Blum and S. Kannan, *Designing Programs that Check their Work,* in STOC 89, pp. 86–97. 59, 60, 62, 64
13. A.De Santis, Y.Desmedt, Y.Frankel, and M.Yung, *How to share a function securely,* in STOC 94, pp. 522–553. 62
14. G. Di Crescenzo, K. Sakurai and M. Yung, *Zero-knowledge proofs of decisions and decision power: new protocols and optimal round-complexity,* in Proc. ICICS'97, pp.18-27. 60, 61, 65
15. G. Di Crescenzo, K. Sakurai and M. Yung, *Result-indistinguishable zero-knowledge proofs: Increased power and constant-round protocols*, in Proc. STACS'98, pp.511-521. 61
16. U. Feige, A. Fiat, and A. Shamir, *Zero-Knowledge Proofs of Identity*, in J. of Cryptology, vol. 1, 1988, pp. 77–94. 60, 61
17. Y.Frankel, P.Gemmell, and M.Yung, *Witness based cryptographic program checking and robust function sharing*, in STOC'96, pp. 499–508. 61, 62
18. L. Fortnow, *The Complexity of Perfect Zero Knowledge*, in STOC 87, pp. 204–209. 70
19. Z. Galil, S. Haber, and M. Yung, *Minimum-Knowledge Interactive Proofs for Decision Problems*, SIAM Journal on Computing, vol. 18, n.4, pp. 711–739 61, 62
20. S. Goldwasser and S. Micali, *Probabilistic Encryption*, Journal of Computer and System Science, vol. 28, n. 2, 1984, pp. 270–299. 59, 66, 69
21. O. Goldreich, S. Micali, and A. Wigderson, *Proofs that Yield Nothing but their Validity* Journal of the ACM, vol. 38, n. 1, 1991, pp. 691–729. 60
22. S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, SIAM J. on Computing, v. 18 (1), Feb. 1989. 60, 62, 66, 67
23. O. Goldreich and E. Petrank, *Quantifying Knowledge Complexity*, in FOCS 91.
24. T. Itoh, M. Hashimoto, and S.Tsujii *A low communication competitive interactive proof system for promised quadratic residuosity*, CRYPTO'93. 66
25. R. Impagliazzo and M. Yung, *Direct Minimum Knowledge Computations*, CRYPTO 87. 69
26. Köbler,J., Schöning,U., and Tóran, J., "The graph isomorphism problem: Its structural complexity," Progress in TCS, *Birkhäuser*,(1993). 59
27. Lund,C., Fortnow,L, Karloff,H., and Nisan,N., "Algebraic methods for interactive proof systems," FOCS'90. 64
28. I. Niven and H. S. Zuckerman, *An Introduction to the Theory of Numbers,* John Wiley and Sons, 1960, New York. 66
29. A. Shamir, *IP=PSPACE*, in FOCS 90, pp. 11–15. (also JACM). 64, 70
30. M.Yung, *Zero-knowledge proofs of computational power*, in Proc. of Eurocrypt'89, LNCS 434, pp.196-207 (1990). 62

Two-Layer Planarization in Graph Drawing

Petra Mutzel and René Weiskircher

Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken
{mutzel,weiski}@mpi-sb.mpg.de

Abstract. We study the two-layer planarization problems that have applications in Automatic Graph Drawing. We are searching for a two-layer planar subgraph of maximum weight in a given two-layer graph. Depending on the number of layers in which the vertices can be permuted freely, that is, zero, one or two, different versions of the problems arise. The latter problem was already investigated in [11] using polyhedral combinatorics. Here, we study the remaining two cases and the relationships between the associated polytopes.

In particular, we investigate the polytope $\mathcal{P}_1$ associated with the two-layer *planarization* problem with one fixed layer. We provide an overview on the relationships between $\mathcal{P}_1$ and the polytope $\mathcal{Q}_1$ associated with the two-layer *crossing minimization* problem with one fixed layer, the linear ordering polytope, the two-layer planarization problem with zero and two layers fixed. We will see that all facet-defining inequalities in $\mathcal{Q}_1$ are also facet-defining for $\mathcal{P}_1$. Furthermore, we give some new classes of facet-defining inequalities and show how the separation problems can be solved. First computational results are presented using a branch-and-cut algorithm. For the case when both layers are fixed, the two-layer planarization problem can be solved in polynomial time by a transformation to the heaviest increasing subsequence problem. Moreover, we give a complete description of the associated polytope $\mathcal{P}_2$, which is useful in our branch-and-cut algorithm for the one-layer fixed case.

1 Introduction

A *bipartite graph* is a graph $G = (A, B, E)$ with vertex sets A and B, called upper and lower layer, and an edge set E connecting a vertex in A with a vertex in B. There are no edges between two vertices in the same layer. A bipartite graph is *two-layer planar* $G = (A, B, E)$ if it can be drawn in such a way that all the vertices in A appear on a line (the upper line), the vertices in B appear on the lower line, and the edges are drawn as straight lines without crossing each other. The difference between a planar bipartite graph and a two-layer planar bipartite graph is obvious. For example, the graph shown in Fig. 1 is a planar bipartite graph, but not a two-layer planar graph.

Depending on the number of layers in which the permutation of the vertices is fixed, different problems arise:

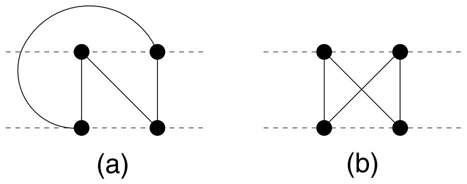


Fig. 1. (a) A planar bipartite graph that is (b) not 2-layer planar

- The permutations π_A and π_B of both layers A and B are fixed: Given a two-layer graph $G = (A, B, E, \pi_A, \pi_B)$ with weights $w_e > 0$ on the edges, the *two-layer planarization problem (2 layers fixed)* is to extract a subgraph $G' = (A, B, F, \pi_A, \pi_B)$, $F \subseteq E$, of maximum weight, i.e., the sum $\sum_{e \in F} w_e$ is maximum, which contains no crossings with respect to the given permutations π_A and π_B.
- The permutation π_A of one layer A is fixed: Given a two-layer graph $G = (A, B, E, \pi_A, \bullet)$ with weights $w_e > 0$ on the edges, the *two-layer planarization problem (1 layer fixed)* is to extract a subgraph $G' = (A, B, F, \pi_A, \bullet)$, $F \subseteq E$, of maximum weight, which contains no crossings with respect to the given permutation π_A of the upper layer.
- Both layers can be permuted: Given a two-layer graph $G = (A, B, E, \bullet, \bullet)$ with weights $w_e > 0$ on the edges, the *two-layer planarization problem (none layer fixed)* is to extract a two-layer planar subgraph $G' = (A, B, F, \bullet, \bullet)$, $F \subseteq E$, of maximum weight.

To our knowledge, only the unweighted ($w_e = 1$ for all $e \in E$) two-layer planarization problems have been considered in the literature so far. Eades and Whitesides [4] showed NP-hardness for the latter two versions of the planarization problem and showed that the two layer fixed version can be solved by transforming it to a longest increasing subsequence problem. The none layer fixed version was first mentioned in [15]. The authors introduced the problem in the context of graph drawing. Recently, the weighted two-layer planarization problem has been attacked, in which the layers are allowed to be permuted freely [11]. The computational results are encouraging.

Directed graphs are widely used to represent structures in many fields such as economics, social sciences, mathematics and computer science. A good visualization of structural information allows the reader to focus on the information content of the diagram.

A common method for drawing directed graphs has been introduced by Sugiyama et al. [14] and Carpano [2]. In the first step, the vertices are partitioned into a set of k layers, and in the second step, the vertices within each layer are permuted in such a way that the number of crossings is small. In practice, this is done layerwise. Keep the permutation of one layer fix while permuting the other one, such that the number of crossings is reduced. We suggest an alternative approach for the second step.

Already for two-layer graphs the straight-line crossing minimization problem is NP-hard [6] even if one layer is fixed [5]. Exact algorithms based on branch and bound have been suggested by various authors (see, e.g., [9]). For $k \geq 2$, a vast amount of heuristics has been published in the literature (see, e.g., [14] and [3]). A new approach is to remove a minimal set of edges such that the remaining k-layer graph can be drawn without edge crossings. In the final drawing, the removed edges are reinserted. Since the insertion of each edge may produce many crossings, the final drawing may be far from an edge-crossing minimal drawing.

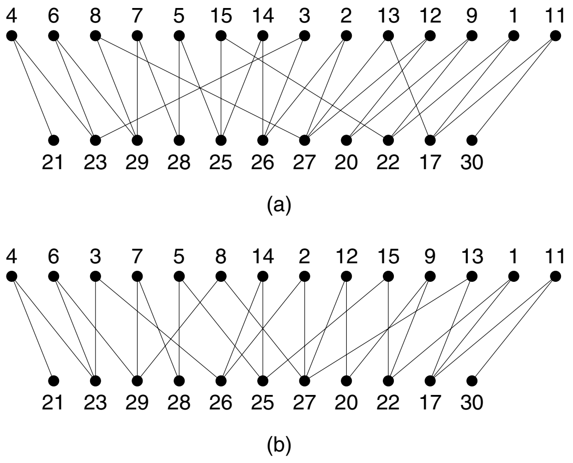


Fig. 2. A graph (a) drawn using k-planarization and (b) drawn with the minimal number of crossings computed by the algorithm in [9]

Figure 2(a) shows a drawing of a graph obtained by two-layer planarization, whereas Fig. 2(b) shows the same graph drawn with the minimal number of edge crossings (using the exact algorithm given in [9]). Although the drawing in Fig. 2(a) has 34 crossings, that is 41% more crossings than the drawing in Fig. 2(b) (24 crossings), the reader will not recognize this fact. This encourages us to study the k-layer planarization problem. We decided to first study the case $k = 2$ in order to learn for the general case $k \geq 3$.

In Sect. 2 we define the polytope $\mathcal{P}_1$ associated with the set of all possible two-layer planar subgraphs with respect to a given fixed permutation π_A. We then point out the relationships to related polytopes. This gives us hints about the structure of $\mathcal{P}_1$. In Sect. 3 we give a complete description of the polytope associated with all two-layer planar subgraphs when both permutations are fixed. This description is useful in the algorithm for solving the two-layer planarization problem (1 layer fixed case). Moreover, it provides a different polynomial time algorithm for solving the two-layer planarization problem (2 layers fixed). In Sect. 4, we investigate the structure of the polytope $\mathcal{P}_1$. We present an irredundant integer linear programming formulation and obtain additional classes of inequalities that tighten the associated LP-relaxation. In particular, besides some new classes of facets, we can show that all facet-defining inequalities of the linear ordering polytope transmit to the new polytope $\mathcal{P}_1$. In order to get practical use out of these inequalities, we have to solve the "separation problem". This question will be addressed in Sect. 5, where we also discuss a branch-and-cut algorithm based on those results. First computational results with a branch-and-cut algorithm are presented in Sect. 6. In this extended abstract we omit the proofs for most of the theorems.

2 The Polytope $\mathcal{P}_1$ and Its Related Polyhedra

Let us consider the two-layer planarization problem with one fixed layer more precisely: Given a two-layer planar graph $G = (A, B, E, \pi_A, \bullet)$ with a fixed permutation π_A of the vertices in A, we are looking for a permutation π_B of the vertices in B, and a subset F of edges in E such that the subgraph $G' = (A, B, F, \pi_A, \pi_B)$ is two-layer planar under the given permutations π_A and π_B of the sets A and B, respectively.

We introduce variables y_{uv} for $1 \leq u < v \leq |B|$ representing the permutation π_B of the vertices in B. That is, $y_{uv} = 1$ iff vertex u is before vertex v in π_B and $y_{uv} = 0$ otherwise. We denote the (row) vector $\bar{y} = (y_{1,2}, y_{1,3}, \ldots, y_L)$ with $L = \binom{B}{2}$. (Vectors are row vectors throughout the paper.) Moreover, we introduce variables x_e for $1 \leq e \leq |E|$ representing the subgraph induced by F. Variable x_e takes value 1 iff $e \in F$ and value 0 otherwise. For any tuple (π_B, F), where π_B is a permutation and $F \subseteq E$, we define an incidence vector $\chi^{(\pi_B,F)} \in \mathrm{R}^{L+|E|}$ with the i-th component $\chi^{(\pi_B,F)}(e_i)$ getting value 1 iff $e_i \in F$ and 0 if $e_i \notin F$ for $i > L$, and the j-th component $\chi^{(\pi_B,F)}(y_{uv})$ getting value 1 if vertex u is before vertex v in π_B and 0 otherwise for $j \leq L$.

Now, we can define the two-layer planar subgraph polytope

$$\mathcal{P}_1 = \mathcal{P}_1(A, B, E, \pi_A, \bullet) = \mathrm{conv}\Big\{\chi^{(\pi_B,F)} \mid \pi_B \text{ is a linear ordering and}$$
$$G' = (A, B, F, \pi_A, \pi_B) \text{ is a two-layer planar subgraph of } G\Big\}$$

as the convex hull of all incidence vectors $\chi^{(\pi_B,F)}$ that represent a two-layer planar subgraph $G' = (A, B, F, \pi_A, \pi_B)$ with respect to the valid orderings π_A and π_B.

For solving the two-layer crossing minimization problem with one fixed layer, we consider the polytope $\mathcal{Q}_1$ (see [9]). Again, we introduce variables $y_{ij} \in \{0, 1\}$ representing the permutation of the vertices in B. The incidence vector $\chi^{\pi_B} \in \mathrm{R}^L$ has the j-th component $\chi^{\pi_B}(y_{uv})$ value 1 if vertex u is before v and 0 otherwise. The polytope

$$\mathcal{Q}_1 = \mathcal{Q}_1(A, B, E, \pi_A, \bullet) = \mathrm{conv}\Big\{\chi^{\pi_B} \mid \pi_B \text{ is a permutation of the vertices in} B\Big\}$$

is identical to the linear ordering polytope that has been studied in [7]. If we denote the points in $\mathcal{P}_1$ by $(\bar{y}, \bar{x})$, where $\bar{y} \in \mathrm{R}^L$, $\bar{x} \in \mathrm{R}^{|E|}$, we have the following relationship between the two polytopes $\mathcal{P}_1$ and $\mathcal{Q}_1$: $\mathcal{Q}_1 \cong P_1 \cap \{\bar{x} = 0\}$. This fact will lead us to investigate the hereditary property of the facet-defining inequalities in $\mathcal{Q}_1$ for $\mathcal{P}_1$.

The polytope $\mathcal{P}_0$ associated with the two-layer planarization problem with two free layers (none fixed) has been introduced in [11]. Again, we denote an incidence vector $\chi^F \in R^E$ having the i-th component $\chi^F(x_e)$ value 1 if $x_e \in F$ and value 0 if $x_e \notin F$.

$$\begin{aligned}\mathcal{P}_0 = \mathcal{P}_0(A, B, E, \bullet, \bullet) = \operatorname{conv}\Big\{\chi^F \mid & \text{ There exist orderings } \pi_A \text{ and } \pi_B \text{ such that} \\ & G' = (A, B, F, \pi_A, \pi_B) \text{ is a two-layer planar subgraph of } G\Big\} = \\ & = \operatorname{conv}\Big\{\chi^F \mid G' = (A, B, F, \bullet, \bullet) \text{ is a two-layer planar subgraph of } G\Big\}\end{aligned}$$

We can use our knowledge of the studied polyhedra $\mathcal{Q}_1$ and $\mathcal{P}_0$ for our investigation of $\mathcal{P}_1$. In particular, all facet-defining inequalities of $\mathcal{Q}_1$ and $\mathcal{P}_0$ are still valid inequalities for $\mathcal{P}_1$. Moreover, we will see that all facet-defining inequalities of $\mathcal{Q}_1$ are still facet-defining for $\mathcal{P}_1$.

Let us consider the two-layer planarization problem when the permutations of both layers are fixed. We define

$$\mathcal{P}_2 = \mathcal{P}_2(A, B, E, \pi_A, \pi_B) = \operatorname{conv}\Big\{\chi^F \mid F \subseteq E \text{ is a two-layer planar subgraph of } G \text{ with respect to the orderings } \pi_A \text{ and } \pi_B\Big\}$$

We have $\mathcal{P}_1 \cap \{\bar{y} = 0\} \supseteq \mathcal{P}_2$. In the following Section we will consider the structure of the polytope $\mathcal{P}_2$.

3 A Complete Description of the Polytope $\mathcal{P}_2$

In this Section we will consider the two-layer planarization problem when both layers are fixed. The set of all two-layer planar subgraph of $G = (A, B, E, \pi_A, \pi_B)$ defines an independence system $\mathcal{I}_P(G) = (E, \{F \mid F \subseteq E \text{ induces a two-layer planar graph}\})$ on E. Let us examine the circuits and the cliques of this independence system. *Circuits* are the minimal dependent sets in $(E, \mathcal{I})$ with respect to set inclusion. An independence set is called *k-regular* if each of its circuits is of size k. The set $F \subseteq E$ is a *clique* of $(E, \mathcal{I})$, if$|F| \geq k$ and all $\binom{|F|}{k}$ k-subsets of F are circuits of $(E, \mathcal{I})$. In [12] it is shown that a maximal clique $F \subseteq E$ in a k-regular independence system $(E, \mathcal{I})$ gives a facet-defining inequality, namely, the *clique inequality*

$$\sum_{e \in F} x_e \leq k - 1, \tag{1}$$

for $P_{\mathcal{I}}$, the polytope associated with $(E, \mathcal{I})$. The set of circuits in our system $\mathcal{I}(G)$ is

$$\mathcal{S} = \{\{(p, v), (q, u)\} \mid \pi_A(p) < \pi_A(q), \pi_B(u) < \pi_B(v), (p, v), (q, u) \in E\}.$$

Hence, $\mathcal{I}(G)$ is a 2-regular independence system. The maximal cliques in $\mathcal{I}(G)$ are the maximal sets of pairwise intersecting edges (with respect to set inclusion). We show, that the associated maximal clique inequalities and the trivial inequalities define the polytope $\mathcal{P}_2$.

Theorem 1. *The maximal clique inequalities of $\mathcal{I}(G)$ together with the inequalities $x_e \leq 1$ that are not contained in any clique and the trivial inequalities $0 \leq x_e$ for $e = 1, \ldots, |E|$, give a complete irredundant description of the two-layer planar subgraph polytope $\mathcal{P}_2$ (both layers fixed).*

Proof. (sketch) To proof the claim, we build a directed graph R with a single source s and a single sink t where every node apart of s and t corresponds to an edge in E. When every node has the capacity given by the corresponding component of a vector x in $[0,1]^{|E|}$, than x belongs to polytope $\mathcal{P}_2$ if and only if there is no path in R from s to t where the sum of the capacities of the nodes is greater than 1. Every path from s to t corresponds to a maximal clique in $(E, \mathcal{I})$ and so a path where the sum of the capacities exceeds one corresponds to a violated clique inequality.

Every vector in $\mathcal{P}_2$ corresponds to a capacity function on the nodes of R such that there is no path from s to t where the sum of the capacities is greater than one. By shifting capacities in R, we can show that for every weighting of the edges in E and for every vector x in $\mathcal{P}_2$, there is another vector x' in $\mathcal{P}_2$ with the property that every component is either 0 or 1 and the sum of the weights of the edges whose nodes in R have capacity 1 is at least as large as the corresponding sum for x. Thus, we have a complete description of $\mathcal{P}_2$.

Since the separation problem for the clique inequalities can be solved in polynomial time (see Sect. 5), this yields a polynomial time algorithm for the two-layer planarization problem via the Ellipsoid method.

There is also a combinatorial algorithm for solving the problem. Eades and Whitesides [4] give a transformation of the unweighted two-layer planarization problem to the longest increasing subsequence problem. A similar transformation to the heaviest increasing subsequence problem works for the weighted version of the problem.

Lemma 1. *By transforming the two-layer planarization problem to an instance of the heaviest increasing subsequence problem, it can be solved in time $\mathrm{O}(|E| \log |E|)$.*

Both theorems are not surprising, since there are similar results for the trace polytope $\mathcal{T}_2$ on two sequences that has been introduced in [13] in the context of multiple sequence alignment. The set of circuits in the independence system $\mathcal{I}_T(G)$ is

$$\begin{aligned} &\mathcal{S} \cup \{\{(p,u),(p,v)\} \mid \pi_B(u) < \pi_B(v), (p,u) \in E, (p,v) \in E\} \\ &\quad \cup \{\{(p,u),(q,u)\} \mid \pi_A(p) < \pi_A(q), (p,u) \in E, (q,u) \in E\}. \end{aligned}$$

In the following, we investigate the relations between the two-layer planar subgraph polytope $\mathcal{P}_2$ and the trace polytope $\mathcal{T}_2$.

Lemma 2. *Let $G = (V, E)$ be a graph. There exist transformations from G to $G' = (V', E')$ and $G'' = (V'', E'')$ with $E' \cong E \cong E''$, $|V'| = |V''| = 2|E|$, and*

$$\mathcal{P}_2(G) \cong \mathcal{P}_2(G') \cong \mathcal{T}_2(G') \text{ and } \mathcal{T}_2(G) \cong \mathcal{T}_2(G'') \cong \mathcal{P}_2(G'').$$

4 The Structure of the Polytope $\mathcal{P}_1$

First, we give an integer linear programming formulation for the two-layer planarization problem with one fixed layer. The notation is based on the previous Section. Let $G = (A, B, E, \pi_A, \bullet)$ be a two-layer graph and let $\bar{w} \in \mathrm{N}^{|E|}$ be the cost vector on the edges. Then, the two-layer planarization problem is to solve

$$\max\{\bar{w}\bar{x}^T \mid (\bar{y}, \bar{x}) \in \mathcal{P}_1, \bar{y} \in \mathrm{R}^L, \bar{x} \in \mathrm{R}^{|E|}\}.$$

We are interested in the integer points of $\mathcal{P}_1$.

Theorem 2. *The integer points of the two-layer planar subgraph polytope $\mathcal{P}_1 = \mathcal{P}_1(A, B, E, \pi_A, \bullet)$ are characterized by the following system of inequalities:*

$$\begin{array}{rcr}
-y_{uv} - y_{vw} + y_{uw} \leq 0 & 1 \leq u < v < w \leq |B| & (2)\\
y_{uv} + y_{vw} - y_{uw} \leq 1 & 1 \leq u < v < w \leq |B| & (3)\\
0 \leq y_{uv} \leq 1 & 1 \leq u < v < w \leq |B| & (4)\\
y_{uv} \textit{ integral} & 1 \leq u < v < w \leq |B| & (5)\\
y_{uv} + x_{(p,u)} + x_{(q,v)} \leq 2 & u < v, \pi_A(q) < \pi_A(p), (p,u), (q,v) \in E & (6)\\
-y_{uv} + x_{(p,u)} + x_{(q,v)} \leq 1 & u < v, \pi_A(p) < \pi_A(q), (p,u), (q,v) \in E & (7)\\
0 \leq x_e \leq 1 & 1 \leq e \leq |E| & (8)\\
x_e \textit{ integral} & 1 \leq e \leq |E| & (9)
\end{array}$$

Proof. The inequalities (2)-(5) require the variables to represent a linear ordering π_B. Inequalities (6)-(9) are responsible for introducing no crossing with respect to the ordering π_B given by the vector $\bar{y}$. In particular, inequalities (6) and (7) link together the subgraph variables $\bar{x}$ and the linear ordering vertices $\bar{y}$. A crossing between two edges (p, u) and (q, v) occurs either if $\pi_A(q) < \pi_A(p)$ and u is before v in the ordering π_B given by $\bar{y}$, or if $\pi_A(p) < \pi_A(q)$ and v is before u in π_B.

Next, we address the question if the description given in Theorem 2 is tight.

Theorem 3. *The description given in Theorem 2 is an irredundant description of the two-layer planar subgraph polytope $\mathcal{P}_1 = \mathcal{P}_1(A, B, E, \pi_A, \bullet)$. In particular, the inequalities (2)-(4) and (6)-(8) are facet-defining for $\mathcal{P}_1$.*

In order to prove the facet-defining property of the inequalities, it is essential to know the dimension of the polytope.

Lemma 3. *The dimension of the two-layer planar subgraph polytope $\mathcal{P}_1 = \mathcal{P}_1(A, B, E, \pi_A, \bullet)$ is $L + |E|$, where $L = \binom{B}{2}$.*

Proof. We know from [7] that the linear ordering polytope is full dimensional. For every ordering of the nodes, a two-layer graph with only one edge is two-layer planar. So we can easily construct a set of $L + |E|$ affinely independent vectors that correspond to two-layer planar graphs.

In Sect. 2 we have seen that $\mathcal{P}_1$ is closely related to the linear ordering polytope $\mathcal{Q}_1$. The following theorem gives us the possibility to use the knowledge of the well-studied polytope $\mathcal{Q}_1$ for $\mathcal{P}_1$.

Theorem 4. *Let $\bar{c}\bar{y}^T \leq c_0$ be a facet-defining inequality of the linear ordering polytope $\mathcal{Q}_1 = \mathcal{Q}_1(A, B, E, \pi_A, \bullet)$. Then $\bar{c}\bar{y}^T \leq c_0$ is also facet-defining for the two-layer planar subgraph polytope $\mathcal{P}_1 = \mathcal{P}_1(A, B, E, \pi_A, \bullet)$.*

For the rest of this Section we will concentrate on new facet-defining inequalities for $\mathcal{P}_1$. Our practical experiments have supported the need for inequalities containing only $\bar{x}$-Variables.

We define a blocker $B = (u, l, r)$ to be a subgraph of $G = (A, B, E, \pi_A, \bullet)$ containing the edges (l, u) and (r, u) with $\pi_A(l) < \pi_A(r)$. We use the notation $x(B) = x_{(l,u)} + x_{(r,u)}$. A blocker forbids certain edges. An edge $e = (p, w)$ *crosses* blocker $B = (u, l, r)$ iff $\pi_A(l) < \pi_A(p) < \pi_A(r)$ and $w \neq u$. This fact leads to *e-blocker inequalities* which are valid and in some cases facet-defining for $\mathcal{P}_1$. Figure 3 shows some examples of configurations leading to these inequalities.

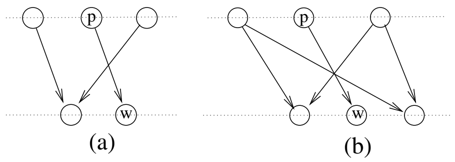


Fig. 3. Examples for support graphs of facet-defining e-blocker inequalities

Theorem 5. *Let $B_1, \ldots, B_k$ be a set of blockers $B_i = (u_i, l_i, r_i)$ with $u_i \neq u_j$ for $i, j = 1, \ldots, k$, $i \neq j$, and let $e = (p, w) \in E$ be an edge which crosses all blockers B_i. Then, the* e-blocker inequality

$$\sum_{i=1}^{k} x(B_i) + x_e \leq k + 1 \tag{10}$$

is valid for $\mathcal{P}_1(A, B, E, \pi_A, \bullet)$. If $l_i = l_j$ and $r_i = r_j$ for $i, j = 1, \ldots, k$, it is facet-defining for $\mathcal{P}_1(A, B, E, \pi_A, \bullet)$ (even for $k = 1$).

5 The Algorithm and Separation Routines

The *separation problem* is to decide for a given vector $\bar{x}$ and a polytope $\mathcal{P}$, whether $\bar{x} \in \mathcal{P}$, and, if $\bar{x} \notin \mathcal{P}$, find a vector $\bar{d}$ and a scalar d_0 such that the inequality $\bar{d}\bar{x}^T \leq d_0$ is valid with respect to $\mathcal{P}$ and $\bar{d}\bar{x}^T > d_0$.

Lemma 4. *The separation problems for the inequalities (2)-(4),(6)-(8), and (10) can be solved in polynomial time.*

For the two-layer planarization problem with one fixed layer, we implemented a branch-and-cut algorithm using the ABACUS-System [10]. Because of space limits, we cannot describe our branch-and-cut algorithm in more detail. We use separation routines for the inequalities given in Lemma 4 in order to get good upper bounds. Moreover, we try to use some information given to us by fractional solutions in order to get good lower bounds.

Our studies of the two-layer fix planarization problem is useful in two ways: For getting good lower bounds, we frequently use the combinatorial algorithm for the two-layer fix version given in Lemma 1. The upper bounds can be improved using the following strategy: In every branching step on variables in $\bar{y}$, we select a variable y_{uv} and set it to either 0 or 1. In the subproblems below this branching node, we have decided on a partial order for the vertices in B. For the partially ordered subsets, we can use the inequalities given by the complete description for $\mathcal{P}_2$ (see Sect. 3). Next, we will see that the separation problem for the inequalities (1) can be solved in polynomial time.

Theorem 6. *For the maximal clique inequalities, the separation problem can be solved in polynomial time by computing at most $|E|$ shortest path problems.*

According to earlier results (e.g., [8]), we can optimize a linear objective function over a polytope in polynomial time if and only if we can solve the separation problem in polynomial time. Hence, Theorem 6 gives us a polynomial time algorithm for solving the two-layer planarization problem when both layers are fixed.

6 Computational Results

To test the performance of our branch-and-cut algorithm for the two-layer planarization problem (1 layer fixed), we worked with the graphs from [1] that are called the *North DAGs*. These directed acyclic graphs have 10 to 100 nodes. We distributed them into sets G_i with i running form 1 to 9 such that the set G_i holds the graphs where the number of nodes is at least $10i$ and at most $10(i+1)-1$. We worked on 12 randomly chosen graphs out of each of the sets G_i. For each of the graphs, we distributed the nodes into pairwise disjoint sets L_j (called *layers*) such that for all edges the start-node is on a layer with smaller index than the end-node. This can be done using topological sorting.

After Inserting some dummy nodes, we get for each graph a number of bipartite graphs that consist of the nodes on two neighboring layers and the edges between these nodes. For a graph with k layers, we get $k-1$ bipartite graphs B_1 to B_{k-1} where B_i consists of the layers L_i and L_{i+1}. We used B_1 as input for our algorithm for solving the two-layer planarization problem (none layer fixed) resulting in a permutation for the layers L_1 and L_2. Then we applied the algorithm for the problem with one fixed layer to the rest of the problems beginning with B_2. Every optimization was stopped after 5 minutes if no optimum solution was found before. The following tabular shows for each set G_i the average optimization time (in seconds) for each layer and the maximum time used in any of the graphs of the set on a Sun Ultra Sparc 2/2x200. In all the 108 graphs we tested, there were only 4 graphs for which a planarization-problem could not be solved to optimality in 5 minutes computation time.

Nodes	10-19	20-29	30-39	40-49	50-59	60-69	70-79	80-89	90-99
Average	0.17	7.07	19.96	1.73	1.37	150.61	12.64	73.07	5.83
Maximum	1.1	17.0	78.53	14.1	27.04	300.16	116.8	300.14	50.16

References

1. G. D. Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vismara. Drawing directed acyclic graphs: An experimental study (preliminary version). Technical Report CS-96-24, Department of Computer Science, Brown University, Oct. 1996. Sun, 13 Jul 1997 18:30:15 GMT. 77
2. M. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-10(11):705–715, 1980. 70
3. P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combinatoria*, 21-A:89–98, 1986. 71
4. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science 131*, pages 361–374, 1994. 70, 74
5. P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 10:379–403, 1994. 71
6. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods*, 4:312–316, 1983. 71
7. M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984. 73, 76
8. M. Grötschel, L. Lovász, and A. Shrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. 77
9. M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *LNCS*, pages 337–348, 1996. 71, 72
10. M. Jünger and S. Thienel. The design of the branch and cut system ABACUS. *Tech. Rep. No. 97.260, Institut für Informatik, Universität zu Köln*, 1997. 77
11. P. Mutzel. An alternative approach for drawing hierarchical graphs. *Proc. Graph Drawing '96, LNCS*, 1997. to appear. 69, 70, 73
12. G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6:48–61, 1973. 73
13. K. Reinert, H. P. Lenhof, P. Mutzel, K. Mehlhorn, and J. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proc. of the 1st Ann. Intern. Conf. on Comp. Molec. Bio. (RECOMB 97)*, Santa Fe, NM, 1997. 74

14. K. Sugiyama, S. Tagawa, and M. Toda. On planarization algorithms of 2-level graphs. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-11:109–125, 1981. 70, 71
15. N. Tomii, Y. Kambayashi, and S. Yajima. On planarization algorithms of 2-level graphs. *Papers of tech. group on electronic computers, IECEJ, EC77-38*, pages 1–12, 1977. 70

Computing Orthogonal Drawings in a Variable Embedding Setting[⋆]
(Extended Abstract)

Walter Didimo[1] and Giuseppe Liotta[2]

[1] Dipartimento di Informatica e Automazione, Università di Roma Tre
via della Vasca Navale 79, 00146 Roma, Italy
didimo@dia.uniroma3.it

[2] Dipartimento di Informatica e Sistemistica, Università di Roma 'La Sapienza'
via Salaria 113, I-00198 Rome, Italy
liotta@dis.uniroma1.it

Abstract. This paper addresses the classical graph drawing problem of designing an algorithm that computes an orthogonal representation with the minimum number of bends, by considering all possible planar embeddings of the graph. While the general problem has been shown to be NP-complete [7], polynomial time algorithms have been devised for graphs whose vertex degree is at most three [5]. We show the first algorithm whose time complexity is exponential only in the number of vertices of degree four of the input graph. This settles a problem left as open in [5]. Our algorithm is further extended to handle graphs with vertices of degree higher than four. The analysis of the algorithm is supported by several experiments on the structure of a large set of input graphs.

1 Introduction and Overview

Graph drawing is concerned with the design of methods for the automatic display of graphs so as to emphasize important features of such graphs and produce aesthetically pleasing visualizations. Various graphic standards have been proposed to draw graphs, each standard being devoted to a specific class of applications. For example, *orthogonal drawings*, i.e. drawings where edges are represented as chains of horizontal and vertical segments, are widely used in applications such as data-base systems (E-R diagrams), software engineering (data-flow diagrams), and circuit design (circuit schematics). An extensive survey on the different graph drawing standards and relative algorithms can be found in [4].

Most graph drawing algorithms are designed to work in a *fixed embedding setting*: they receive as input a graph G with information about its topology and produce as output a drawing of G that preserves such information. For a planar

⋆ Research supported in part by the ESPRIT LTR Project no. 20244 - ALCOM-IT and by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD."

graph, such topological information is typically represented by its *planar embedding*, i.e. the circular ordering of the edges incident on the vertices. If the input graph is not planar, the first step of the algorithm is typically a *planarization step* that determines a planar embedding of the graph, possibly adding dummy vertices for crossings. The planarization step is then followed by a *representation step*, that computes a drawing with the given embedding and removes the dummy vertices. Notice that the choice of a planar embedding can deeply affect the output of the drawing algorithm. Thus, it naturally rises the problem of designing algorithms that work in a *variable embedding setting*, i.e. algorithms that are allowed to change the planar embedding of the input graph, in order to optimize the output with respect to a given set of aesthetic requirements (see, e.g. [1,7,2,5]).
In this paper we study the problem of computing orthogonal drawings with the minimum number of bends in a variable embedding setting. Orthogonal drawings are a classical field of investigation in graph drawing. A very limited sample of papers on orthogonal drawings includes [13,14,11,3,12]. In [7] it is shown that the problem of computing an orthogonal drawing with the minimum number of bends is NP-complete. On the other hand, polynomial time algorithms for series-parallel graphs and 3-planar graphs are proposed in [5]. The algorithm in [2] computes an orthogonal drawing with the minimum number of bends for biconnected 4-planar graphs, by considering the set of all planar embeddings of the graph with a branch-and-bound technique. Since the number of such embeddings is in general exponential with the number of vertices of the graph, the algorithm can require exponential time. The research described in this paper starts from the following observation: based on the existence of a polynomial time solution for 3-planar graphs, it would be desirable to have an algorithm whose complexity is exponential only in the number of vertices of degree 4. This can be of particular interest if such number is a small fraction of the total number of vertices. Consider for example the graph of Figure 1, that has 45 vertices, only one vertex of degree 4, more than 2^{21} different planar embeddings, and whose orthogonal drawing with the minimum number of bends can be computed in polynomial time by an easy variant of the algorithm in [5]. Devising a $O(c^{n_4}p(n))$ algorithm for computing orthogonal drawings with the minimum number of bends in a variable embedding setting is also mentioned as an open problem in [5]. Here, c is a constant, n_4 is the number of vertices of degree 4 of the input graph, and $p(n)$ is a polynomial in the total number of vertices of the graph. In the paper, we exploit a vertex expansion technique to devise a $O(6^{n_4}n^4 \log n)$-time algorithm, that receives as input a biconnected 4-planar graph G and computes an orthogonal drawing of G with the minimum number of bends. The analysis of the algorithm is supported by several experiments on the structure of a large set of input graphs. The algorithm is further extended to handle biconnected planar graphs with vertices of degree higher than 4. Ideas related to those that underly our vertex expansion technique can be found in [9]. For lack of space, proofs have been omitted in this extended abstract and will appear in the full version of the paper (see also `www.dia.uniroma3.it/∼didimo/papers`).

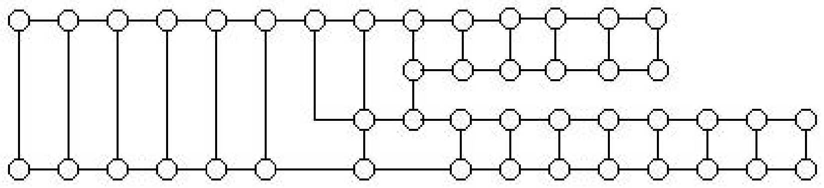


Fig. 1. An orthogonal drawing of a graph with 45 vertices, 1 vertex of degree 4, and more than 2^{21} different planar embeddings. The graph has been drawn with the GDToolkit system (`www.dia.uniroma3.it/people/gdb/wp12/`).

2 Preliminaries

We assume familiarity with planarity and connectivity of graphs [10]. Since we consider only planar graphs, we often use the term *embedding* instead of *planar embedding*. A graph is said to be *k-planar* if it is planar and its vertices have degree at most k, where k is an integer constant. The degree of a vertex v is denoted as $deg(v)$.

Let G be a biconnected planar graph. A *split pair* of G is either a separation pair of G or a pair of adjacent vertices. A *split component* of a split pair $\{u, w\}$ is either an edge (u, w) or a maximal subgraph G_{uw} of G such that $\{u, w\}$ is not a split pair of G_{uw}. Vertices u and w are the *poles* of the split component. Let (s, t) be an edge of G called *reference edge*. An *SPQ^*R-tree* $\mathcal{T}$ of G (a simple variation of the $SPQR$-trees introduced in [6]) is a tree describing a recursive decomposition of G with respect to its split pairs, and it is used to synthetically represent all the embeddings of G with (s, t) on the external face (we always assume, unless stated otherwise, that (s, t) is on the external face). Nodes of $\mathcal{T}$ are of four types: S, P, Q^* and R. Each node μ has an associated graph called *skeleton* of μ and denoted by $skeleton(\mu)$. Starting from the reference edge, $\mathcal{T}$ is recursively defined as follows: *Chain case*: if G consists of a simple path from s to t then $\mathcal{T}$ is a single Q^*-node μ whose skeleton is G itself. *Series case*: if G is 1-connected, let $c_1,\ldots, c_{k-1}$ $(k \geq 2)$ be the cutvertices of G such that no cutvertex has degree less than three; $c_1,\ldots, c_{k-1}$ partition G into graphs $G_1,\ldots, G_k$. The root of $\mathcal{T}$ is an S-node μ. Graph $skeleton(\mu)$ is the chain $e_1, \ldots e_k$, where edge e_i goes from c_{i-1} to c_i, $c_0 = s$ and $c_k = t$. *Parallel case*: if s and t are a split pair for G with split components $G_1,\ldots, G_k$ $(k \geq 2)$, the root of $\mathcal{T}$ is a P-node μ. Graph $skeleton(\mu)$ consists of k parallel edges from s to t, denoted $e_1, \ldots, e_k$. *Rigid case*: if none of the above cases applies, let $\{s_1, t_1\}$,...,$\{s_k, t_k\}$ be the maximal split pairs of G $(k \leq 1)$, and for $i = 1..k$, let G_i be the union of all the split components of $\{s_i, t_i\}$. The root of $\mathcal{T}$ is an R-node μ. Graph $skeleton(\mu)$ is obtained from G by replacing each subgraph G_i with edge e_i from s_i to t_i. We call *pertinent graph* of μ the graph whose decomposition tree is the subtree rooted at μ. Also, the *virtual edge* of μ is the edge representing the pertinent graph of μ in the skeleton of its parent. An *R-component* (*S-component*, *P-*

component) of G is the pertinent graph of an R-node (S-node, P-node) of a decomposition tree of G.
Let Ψ be a planar embedding of G and let f be a face of G. f is a *rigid face* if all its edges belong to the skeleton of an R-node of $\mathcal{T}$. An immediate consequence of the properties of S, P, Q^*-trees is the following.

Property 1. If f is a rigid face in a planar embedding of G, then the edges of f form a rigid face in any planar embedding of G.

A planar drawing of a planar graph G such that all edges of G are mapped to polygonal chains of horizontal and vertical segments is an *orthogonal drawing* of G. A planar graph has an orthogonal drawing if and only if it is 4-planar. Two orthogonal drawings Γ and Γ' of G, preserving the same embedding of G, are *shape equivalent* if for each vertex v, consecutive edges in the adjacency list of v form the same angle in the two drawings, and for each edge (u, v) following from u to v the polygonal chain representing (u, v), we have the same (possibly empty) sequence of left and right turns in the two drawings. An *orthogonal representation* H of G is a class of shape equivalent orthogonal drawings of G. We say that H *preserves* the embedding of G when the shape equivalent orthogonal drawings in the class H preserve the embedding of G. We also say that H is *optimal within the given embedding* if it has the minimum number of bends among all orthogonal representations of G that preserve its embedding. Finally, H is *optimal*, if it has the minimum number of bends over all the possible (not necessarily embedding preserving) orthogonal representations of G. Turns on a polygonal chain of H are called *bends*. We denote by $b(H)$ the total number of bends of H.

Let H be an orthogonal representation of a 3-planar graph and let f be an internal face of H, such that f consists of 4 vertices and has no bends (i.e. f is a rectangle with a vertex at each corner). Let v be a vertex of f such that $deg(v) = 3$ and let e' and e'' be the two edges of f that share v. Edge e' (e'') is the *ingoing* (*outgoing*) edge of v in f if the triplet e', v, e'' is encountered in this order when walking around f in clockwise direction. Face f is said to be a *counterclockwise* (*clockwise*) face if for every vertex v of f such that deg(v)=3, the edge e of H incident on v and not belonging to f forms a $\pi/2$ angle with the ingoing (outgoing) edge of v (see Figure 2).

3 Expanding Graphs and Representations

Let G be a biconnected 4-planar graph with a given planar embedding Ψ and let Γ be a planar drawing of G that preserves Ψ. Let v be a vertex of Γ such that $deg(v) = 4$ and let e_1, e_2, e_3, e_4 be the edges incident on v in circular clockwise order. We construct a planar drawing from Γ as follows: (1) v is replaced in Γ by a simple cycle $\tilde{f}_v$, called *expansion* of v, that consists of vertices $\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, \tilde{v}_4$ in clockwise order around $\tilde{f}_v$. (2) each edge $e_i = (u_i, v)$ of Γ is replaced by an edge $\tilde{e}_i = (u_i, \tilde{v}_i)$, $i = 1, \ldots, 4$. Let $\widetilde{\Gamma}$ be a drawing obtained by recursively applying operations (1) and (2) to Γ, for each vertex of Γ that has degree larger than

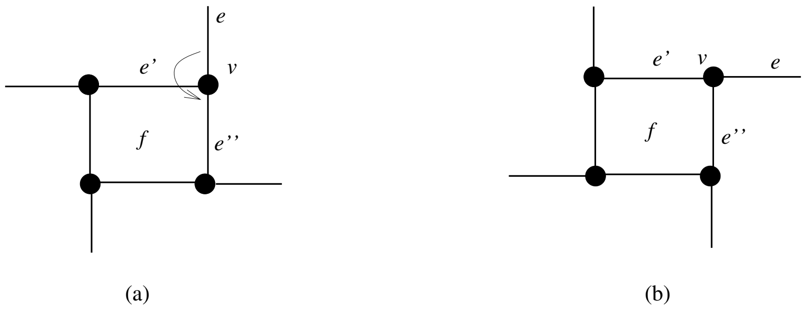


Fig. 2. (a) A counterclockwise face. (b) A clockwise face.

three. See, for example, Figure 3 (b). We call the planar graph represented by $\widetilde{\Gamma}$ the *expanded graph of* G and denote it as $\widetilde{G}$. It is univocally determined by Ψ.

Property 2. The expanded graph of G is a 3-planar graph.

Property 3. The expanded graph of G has $n+3n_4$ vertices, where n and n_4 denote the number of vertices and the number of degree 4 vertices of G, respectively.

The class of drawings of $\widetilde{G}$ that are equivalent to $\widetilde{\Gamma}$ defines an embedding of $\widetilde{G}$ that is called the *expanded embedding of* Ψ.

Property 4. In the expanded embedding of Ψ an expansion of a vertex of G is an internal face.

A face that is the expansion of a vertex of G is also called *expanded face.*

Theorem 1. *Let G be an embedded biconnected 4-planar graph and let $\widetilde{G}$ be an expanded graph of G. Let v be a vertex of G such that $deg(v) = 4$ and let $\tilde{f}_v$ be the expansion of v in $\widetilde{G}$. Cycle $\tilde{f}_v$ is a rigid face in any planar embedding of $\widetilde{G}$.*

Let G be an embedded 4-planar graph, let Ψ be the embedding of G, let $\widetilde{G}$ be the expanded graph of G, and let $\widetilde{\Psi}$ be the embedding of $\widetilde{G}$. We now study the relationship between the set of orthogonal representations of G that preserve Ψ and a particular set of orthogonal representations of $\widetilde{G}$. Namely, we consider the set $\mathcal{H}_{\widetilde{\Psi}}$ of orthogonal representations of $\widetilde{G}$ such that: each orthogonal representation $\widetilde{H}$ of $\mathcal{H}_{\widetilde{\Psi}}$ preserves $\widetilde{\Psi}$, and each face of $\widetilde{G}$ resulting from the expansion of a vertex of G is represented in $\widetilde{H}$ as a counterclockwise face. An element of $\mathcal{H}_{\widetilde{\Psi}}$ is called *counterclockwise orthogonal representation* of $\widetilde{G}$. Let $\mathcal{H}_{\Psi}$ be the set of all orthogonal representations of G that preserve Ψ.

Lemma 1. *For any orthogonal representation $H \in \mathcal{H}_{\Psi}$ there exists an orthogonal representation $\widetilde{H} \in \mathcal{H}_{\widetilde{\Psi}}$ such that $b(H) = b(\widetilde{H})$. Also, for any orthogonal representation $\widetilde{H} \in \mathcal{H}_{\widetilde{\Psi}}$ there exists an orthogonal representation $H \in \mathcal{H}_G$ such that $b(\widetilde{H}) = b(H)$.*

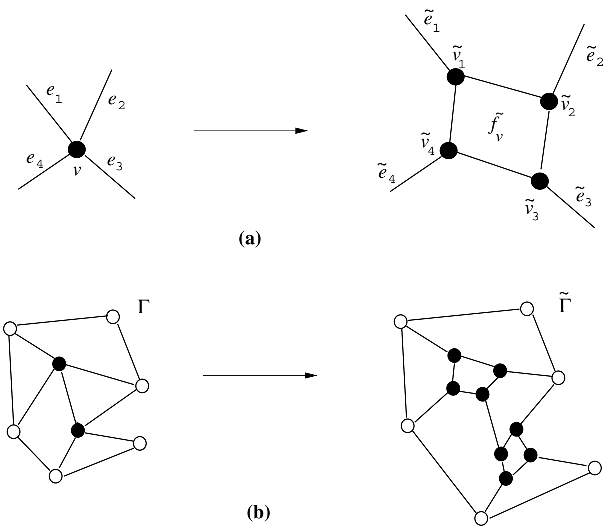


Fig. 3. (a) a vertex v and its expansion $\tilde{f}_v$; (b) a drawing Γ and a drawing $\widetilde{\Gamma}$ obtained by recursively applying operations (1) and (2)

Theorem 2. *Let G be an embedded 4-planar graph, let Ψ be the embedding of G, let $\widetilde{G}$ be the expanded graph of G, and let $\widetilde{\Psi}$ be the embedding of $\widetilde{G}$. There exists a bijection $\mathcal{E}$ between the set $\mathcal{H}_\Psi$ of orthogonal representations of G and the set $\mathcal{H}_{\widetilde{\Psi}}$ of counterclockwise orthogonal representations of $\widetilde{G}$. Also, for any $H \in \mathcal{H}_\Psi$ we have that $b(H) = b(\mathcal{E}(H))$.*

In the rest, for an orthogonal representation $H \in \mathcal{H}_\Psi$ we call its image $\widetilde{H} = \mathcal{E}(H)$ the *expansion of H*. Also, $H = \mathcal{E}^{-1}(\widetilde{H})$ is the *contraction of $\widetilde{H}$*.

4 Optimal Orthogonal Drawings

Our general strategy for computing optimal orthogonal drawings consists of two main steps. The input is a biconnected 4-planar graph G and the output is an optimal orthogonal drawing of G. We use SPQ^*R-trees to handle the embeddings of G. Since a decomposition tree of G implicitly represents the embeddings of G with a reference edge on the external face, for each choice of a reference edge e we compute an optimal orthogonal drawing Γ of G, such that e is on the external face (`Computation Step`). After the execution of the `Computation Step` on each possible choice of the reference edge, we select the orthogonal drawing with the minimum number of bends (`Selection Step`). Since there are $O(n)$ possible choices of the reference edge, the overall time complexity of a drawing algorithm based on the above strategy is $O(n \times t(n))$, where $t(n)$ is the time complexity of the `Computation Step`.

Now, we focus on the `Computation Step`. From the assumption that G is 4-planar and using the properties of the decomposition trees, the following lemma gives an upper bound on the number of embeddings of G with e on the external face. We denote with $\mathcal{P}_e$ the set of such embeddings.

Lemma 2. *$|\mathcal{P}_e| \leq 6^n$, where n is the number of vertices of G.*

Based on Lemma 2, a brute-force approach to the `Computation Step` is as follows: (i)For any embedding Ψ of $\mathcal{P}_e$ compute an optimal orthogonal drawing that preserves Ψ by means of the $O(n^2 \log n)$ algorithm in [13]. (ii)Choose the drawing with the minimum number of bends among the computed ones. Hence a `Computation Step` based on this approach requires $O(6^n n^2 \log n)$ time. The bound can be further reduced to $O(6^n n^{\frac{11}{4}} \sqrt{logn})$ time with the technique presented in [8]. However, as also pointed out in Section 1, the `Computation Step` can be executed in polynomial time if G is a 3-planar graph [5]. Thus, we aim at devising an alternative strategy for such step, that is exponential in the number of degree 4 vertices of G. Our idea is to execute the `Computation Step` as follows: **Task (i)**: The embeddings in $\mathcal{P}_e$ are grouped into $O(c^{n_4})$ equivalence classes. **Task (ii)**: For each class of equivalent embeddings, an orthogonal drawing with the minimum number of bends is computed in polynomial time. **Task (iii)**: Among the $O(c^{n_4})$ computed orthogonal drawings, the cheapest one is chosen.

Regarding Task (i), we need to define a new notion of equivalence between the embeddings of G. Let Ψ_1 and Ψ_2 be two embeddings in $\mathcal{P}_e$. Ψ_1 and Ψ_2 are 4-*equivalent* if they have the same circular counterclockwise ordering of the edges around each vertex v of G such that $deg(v) = 4$. Hence, $\mathcal{P}_e$ can be partitioned into equivalence classes such that any two embeddings in the same class are 4-equivalent. A 4-*embedding* is a class of 4-equivalent embeddings in $\mathcal{P}_e$. We denote with $\mathcal{P}_e^{(4)}$ the set of 4-embeddings of $\mathcal{P}_e$.

Lemma 3. *$|\mathcal{P}_e^{(4)}| \leq 6^{n_4}$, where n_4 is the number of vertices of degree 4 of G.*

The following lemma is a consequence of Lemma 3 and of a variant of the algorithm in [2] that generates all embeddings of a given graph.

Lemma 4. *Let G be a biconnected 4-planar graph with a given reference edge e and n_4 vertices of degree 4. The set of 4-equivalent embeddings $\mathcal{P}_e^{(4)}$ can be computed in $O(6^{n_4})$ time.*

For a given reference edge e of G, let $\Psi_e^{(4)}$ be a 4-embedding in $\mathcal{P}_e^{(4)}$. An orthogonal representation of G that is optimal within the embeddings in $\Psi_e^{(4)}$ is called *optimal within the* 4-*embedding* $\Psi_e^{(4)}$, and is denoted as $H_e^{(4)}$. Task (ii) is accomplished by computing for each $\Psi_e^{(4)}$ a corresponding $H_e^{(4)}$. To do this in polynomial time, we exploit the expansion theory of Section 3 as follows. We consider the expanded graph $\widetilde{G}$ of G induced by an embedding in $\Psi_e^{(4)}$. By Property 2 $\widetilde{G}$ is 3-planar and by Property 3 it has $O(n)$ vertices. In [5] it is shown an algorithm that computes an optimal orthogonal representation of a 3-planar graph in polynomial time. First, the graph is decomposed in its S-, P-, and R-components by means of an SPQ^*R-tree. Then, for each S-, P-, and R-component, a suitable set of orthogonal representations is computed, and such representations are combined to compose an optimal one. In particular, for each R-node the set of orthogonal representations is computed by solving a minimum cost flow problem on a flow network associated to the skeleton of the node. Since by Theorem 1, each expanded face in $\widetilde{G}$ is rigid. By adding constraints to the minmum cost flow problem studied in [5], a variant of the that algorithm can be devised to compute an orthogonal representation $\widetilde{H}$ of $\widetilde{G}$ such that: **(a)** Its embedding is the expanded embedding of an element in $\Psi_e^{(4)}$. **(b)** Its expanded faces are counterclockwise. **(c)** It has the minimum number of bends among all orthogonal representations of $\widetilde{G}$ satisfying (a) and (b).

By means of techniques similar to those presented in [5] the following can be proved.

Lemma 5. *$\widetilde{H}$ can be computed in $O(n^3 \log n)$ time.*

By applying Theorem 2 we have the following result.

Lemma 6. *The contraction of $\widetilde{H}$ is an orthogonal representation of G optimal within $\Psi_e^{(4)}$.*

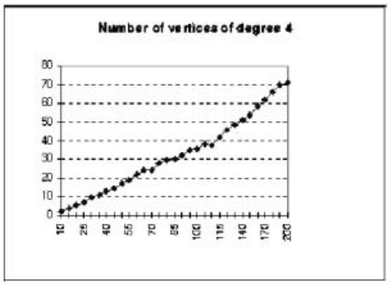
Lemmas 5 and 6 give rise to the following.

Theorem 3. *Let G be a biconnected 4-planar graph with n vertices and let n_4 be the number of vertices of G that have degree 4. There exists an algorithm that computes an orthogonal drawing of G with the minimum number of bends in $O(6^{n_4} n^4 \log n)$ time.*

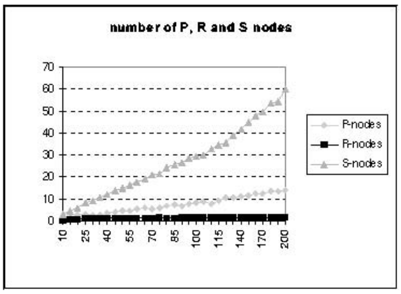
The algorithm described above can be extended to compute optimal orthogonal drawings of graphs with high degree. Namely, the notion of expanded graphs can be readily defined also to graphs with high degree vertices. In this case, the expanded face $\tilde{f}_v$ of a vertex v such that $deg(v) = k$ consists of k vertices. Vertex v is then represented in the computed optimal orthogonal drawing as a rectangular face that actually corresponds to $\tilde{f}_v$. By exploiting the 3-planarity of the expanded graph, we can state the following.

Theorem 4. *Let G be a biconnected k-planar graph with n vertices, where $k \geq 4$ is the maximum vertex degree. Let $\tilde{n}$ be the number of vertices of G that have degree larger or equal to 4. There exists an algorithm that computes an orthogonal drawing of G with the minimum number of bends in $O((k-1)!^{\tilde{n}}$ $((k-1)n)^4 \log((k-1)n)$ time.*

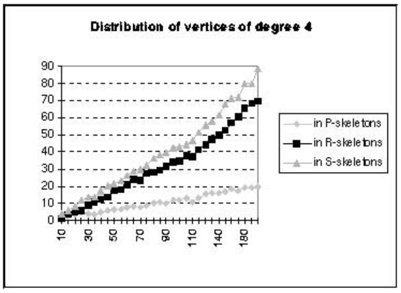
We performed several experiments in order to better understand the behavior of our algorithm on a large set of graphs. We considered a test suite of more than 2000 graphs, with number of vertices in the range 10–200, and having a number of 4 degree vertices linear with the total number of vertices (Figure 4 (a)).



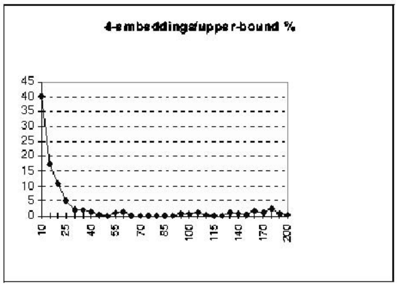
(a)



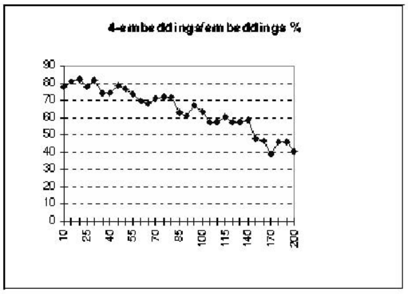
(b)



(c)



(d)



(e)

Fig. 4. Charts of the experiments: the x-axis represents the number of vertices.

The graphs have been generated by a technique that has been already exploited for other experimental studies on orthogonal drawings [2]. The implementation uses the `GDToolkit` library (`www.dia.uniroma3.it/people/gdb/wp12/`). We observed the quantity $\frac{|\mathcal{P}_e^{(4)}|}{6^{n_4}}$ (Figure 4 (d)), in order to measure in practice the factor c^{n_4} affecting the overall time complexity of the algorithm. Also, we measured the quantity $\frac{|\mathcal{P}_e^{(4)}|}{|\mathcal{P}_e|}$ (Figure 4 (e)), in order to estimate the effectiveness of our algorithm with respect to a brute force approach.

Figure 4 (b) shows the structure of the graphs in terms of the number of their distinct S-, P- and R-nodes of the decomposition tree. The distribution of vertices of degree 4 in such components is depicted in Figure 4 (c).

Acknowledgments

We thank Giuseppe Di Battista for his suggestions and encouragement.

References

1. P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. How to draw a series-parallel digraph. *Internat. J. Comput. Geom. Appl.*, 4:385–402, 1994. 80
2. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 331–344. Springer-Verlag, 1997. 80, 85, 88
3. T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9:159–180, 1998. 80
4. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994. 79
5. G. Di Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM Journal on Computing.* to appear. 79, 80, 85, 86
6. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25:956–997, 1996. 81
7. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. Submitted to *SIAM Journal on Computing*, 1995. 79, 80
8. A. Garg and R. Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. C. North, editor, *Graph Drawing (Proc. GD '96)*, Lecture Notes Comput. Sci. Springer-Verlag, 1997. 85
9. P. Mutzel. *The Maximum Planar Subgraph Problem.* Ph.D. thesis, Köln Univ., Köln, 1994. 80
10. T. Nishizeki and N. Chiba. Planar graphs: Theory and algorithms. *Ann. Discrete Math.*, 32, 1988. 81
11. A. Papakostas and I. G. Tollis. Improved algorithms and bounds for orthogonal drawings. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 40–51. Springer-Verlag, 1995. 80

12. M. S. Rahman, S. Nakano, and T. Nishizeki. A linear algorithm for optimal orthogonal drawings of triconnected cubic plane graphs. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 99–110. Springer-Verlag, 1998. 80
13. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. 80, 85
14. R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989. 80

Dynamic Grid Embedding with Few Bends and Changes

Ulrik Brandes and Dorothea Wagner

Faculty of Mathematics and Computer Science
University of Konstanz
D–78457 Konstanz, Germany
{Ulrik.Brandes,Dorothea.Wagner}@uni-konstanz.de

Abstract. In orthogonal graph drawing, edges are represented by sequences of horizontal and vertical straight line segments. For graphs of degree at most four, this can be achieved by embedding the graph in a grid. The number of bends displayed is an important criterion for layout quality. A well-known algorithm of Tamassia efficiently embeds a planar graph with fixed combinatorial embedding and vertex degree at most four in the grid such that the number of bends is minimum [23].
When given a dynamic graph, i.e. a graph that changes over time, one has to take into account not only the static criteria of layout quality, but also the effort users spent to regain familiarity with the layout. Therefore, consecutive layouts should compromize between quality and stability. We here extend Tamassia's layout model to dynamic graphs in a way that allows to specify the relative importance of the number of bends vs. the number of changes between consecutive layouts. We also show that optimal layouts in the dynamic model can be computed efficiently by means that are very similar to the static model, namely by solving a minimum cost flow problem in a suitably defined network.

1 Introduction

One way of reducing perceptual complexity in visualized networks is to represent all edges by alternating sequences of horizontal and vertical straight line segments. Technical requirements, as in VLSI layout, may be another reason for choosing this form of representation. Along with the generally increasing interest in graph drawing (see [11] for a survey), significant attention has been devoted to orthogonal graph layout. Many efficient algorithms have been designed, and many properties of orthogonal layouts (such as the number of bends, the number of crossings, or the area needed) have been analyzed [23,4,2,20,6, and many more].

For planar graphs with vertex degree at most four, Tamassia [23] gave an efficient algorithm computing a grid layout with the minimum number of bends preserving a given combinatorial embedding of the graph. If the embedding is not fixed in advance, bend minimization becomes $\mathcal{NP}$-complete [16]. The algorithm is well-known both because of its conceptual elegance and because the number of bends seems to be an important criterion for layout quality [22]. It is based on the correspondence of flow in an associated network and angles in the embedding. The connection between angles in straight-line representations and values of certain linear programs was investigated by a number of authors [25,18,12,15], but is not one-to-one in general. Tamassia's algorithm has been extended to graphs of higher degree [14], and it builds the core of an algorithm for non-planar graphs of arbitrary degree [24].

In many settings such as user interaction, software visualization, animation of graph algorithms, or graph queries, it is required to deal with dynamic graphs, i.e. graphs that change over time. When a user analyzes a graph visually, he or she builds a mental map that ought not change dramatically when the graph is modified [13]. There are several approaches to dynamic layout [10,7,19], but most research on orthogonal layout has focused on incremental updates, in which only creation of new vertices and edges is allowed [21,5]. InteractiveGiotto [3] is a system allowing arbitrary updates, but the layout of remaining portions of the graph is fixed, which generally increases the number of bends needed. Here, we apply the Bayesian framework of [8] to obtain a dynamic model which forms an explicit and controllable compromise between layout stability and layout quality. Interestingly, the corresponding optimization problem can still be solved efficiently by network flow techniques.

This paper is organized as follows: In Sect. 2, we review the bend minimum grid embedding of [23]. After deriving an objective function for dynamic layout in Sect. 3, we show how to compute a layout accordingly in Sect. 4. Forms of user interaction are discussed briefly in Sect. 5, and examples are given in Sect. 6.

2 Static Bend Minimum Layout

A graph is said to be *4-planar*, if it is planar and has vertex degree at most four. For the orthogonal layout of connected 4-planar graphs with fixed combinatorial embedding, Tamassia [23] generates a flow network and computes a minimum cost flow corresponding to an orthogonal representation with the minimum number of bends preserving the embedding. In this section, we review briefly the concept of orthogonal representation and the construction of the flow network associated to the graph.

Let G be an embedded, connected, 4-planar graph. As shown in Fig. 1, an *orthogonal representation* $H(G)$ of $G = (V, E)$ consists of circular lists $H_f = [\,(e_0, s_0, a_0), \ldots, (e_{d_G(f)-1}, s_{d_G(f)-1}, a_{d_G(f)-1})\,]$ for each face f in G. Each triple (e_i, s_i, a_i) of list H_f consists of an edge $e_i \in E$, a string $s_i \in \{0,1\}^*$, and an integer $a_i \in \{90, 180, 270, 360\}$, such that $e_1, \ldots, e_{d_G(f)-1}$ is a clockwise (counterclockwise, if f is the outer face) traversal of the edges incident to f. Note that edges incident to vertices of degree one appear twice in this traversal. In s_i, 0's and 1's represent 90 and 270 degree bends, respectively, on the right side of e_i in the traversal. Finally, a_i is the size of the angle between e_i and $e_{i+1 \bmod d_G(f)}$.

Lemma 1 ([23]). *If a b-bend orthogonal representation of an n-vertex graph is given, a corresponding grid embedding can be computed in time $\mathcal{O}(n + b)$.*

A flow network $N = (W, A; b, l, u, c)$ consists of a directed graph (W, A) and functions b (*supply/demand*), l (*lower capacity*), u (*upper capacity*), and c (*cost*) defined on the set of arcs. We here assume that all four take integer values and that lower capacities are nonnegative. To avoid confusion with the vertices of the planar graph to be embedded, the elements of W are called *nodes*. A *flow* $x = (x_a)_{a \in A}$ with *cost* $\sum_{a \in A} c(a) \cdot x_a$ is an integer valued vector such that

$$\begin{aligned} b(w) + \sum_{(w',w) \in A} x_{(w',w)} - \sum_{(w,w') \in A} x_{(w,w')} &= 0 \quad \text{for all } w \in W \\ l(a) \leq x_a \leq u(a) & \quad\quad \text{for all } a \in A \end{aligned}$$

For the existence of a feasible flow, it is necessary that the total mass balance constraint, $\sum_{w \in W} b(w) = 0$, is satisfied. A vector x satisfying $\sum_{(w',w) \in A} x_{(w',w)} = \sum_{(w,w') \in A} x_{(w,w')}$, $w \in W$, and $l(a) \leq x_a \leq u(a)$, $a \in A$, is called a *circulation*. See [1] for a comprehensive treatment of network flows.

The flow network $N(G)$ used to compute some $H(G)$ is constructed such that each unit of flow corresponds to a 90 degree angle in $H(G)$. Therefore, the node set $W = W_V \cup W_F$ of $N(G)$ consists of all vertices $v \in V$ and faces $f \in F$ of G, i.e. $W_V = V$ and $W_F = F$. Likewise, two types of arcs make up $A = A_V \cup A_F$: For every occurrence of a vertex $v \in V$ in the traversal of a face $f \in F$ there is an arc $(v, f) \in A_V$. For every edge incident to neighboring faces f and g there are arcs $(f, g), (g, f) \in A_F$. Every vertex node $v \in W_V$ has supply $b(v) = 4$, while face nodes $f \in W_F$ have demand

$$b(f) = \begin{cases} -(2 \cdot d_G(f) - 4) & \text{if } f \text{ is an internal face} \\ -(2 \cdot d_G(f) + 4) & \text{if } f \text{ is the external face} \end{cases}$$

By Euler's formula, $\sum_{v \in W_V} b(v) + \sum_{f \in W_F} b(f) = 0$, i.e. the total mass balance constraint is satisfied. The lower and upper capacity of a vertex-face arc $(v, f) \in A_V$ are $l(v, f) = 1$ and $u(v, f) = 4$, respectively, while a face-face arc $(f, g) \in A_F$ has $l(f, g) = 0$ and infinite upper capacity. Arcs in A_V have zero cost, while there is an integer cost $\gamma \geq 0$ on arcs in A_F. See Fig. 1.

Theorem 1 ([23]). *If $\gamma > 0$, there is a 1-1 correspondence between feasible flows in $N(G)$ and orthogonal representations of G (up to the ordering of bends on each edge). Moreover, the total cost of a flow in $N(G)$ corresponding to an orthogonal representation with b bends equals $\gamma \cdot b$.*

3 Dynamic Layout Model

In the remainder of this paper, let G, $G^{(1)} = (V^{(1)}, A^{(1)})$, and $G^{(2)} = (V^{(2)}, A^{(2)})$ be any triple of connected, embedded, 4-planar graphs. $G^{(2)}$ succeeds $G^{(1)}$ in a sequence of graphs and is to be laid out with few bends and changes with respect to a given layout of $G^{(1)}$.

In order to obtain a suitable dynamic layout model we first formulate Tamassia's static model in a slightly different way: Given a connected, embedded, 4-planar graph G and its associated flow network $N(G)$, the range of variables x_a,

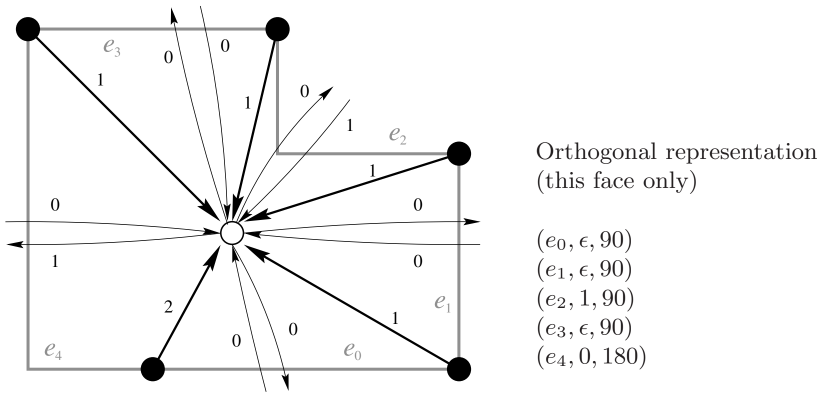


Fig. 1. Part of an embedded 4-planar graph G and its associated flow network $N(G)$. Vertex nodes $v \in W$ have $b(v) = 4$, while the face node $f \in W$ has $b(f) = -(2 \cdot 5 - 4) = -6$. Thick arcs have capacity $[1, 4]$ and cost zero, the others have capacity $[0, \infty)$ and cost γ. All arcs are labeled with values of a feasible flow, and the (*grey*) edges of G are routed according to the corresponding orthogonal representation

$a \in A$, is $\{l(a), \ldots, u(a)\} = \mathcal{X}_a$. Let $\mathcal{X}$ be the cartesian product of all $\mathcal{X}_a$, $a \in A$, and $\hat{\mathcal{X}} \subseteq \mathcal{X}$ be the set of feasible flow vectors. By Theorem 1, each $x \in \hat{\mathcal{X}}$ represents an orthogonal representation of G. The objective function of the minimum cost flow problem is $U(x) = \sum_{a \in A} c(a) \cdot x_a = \sum_{a \in A_F} \gamma \cdot x_a$, where γ is a nonnegative integer. A random variable X with distribution

$$P(X = x) = \begin{cases} \frac{1}{Z}\, e^{-U(x)} & \text{if } x \in \hat{\mathcal{X}} \\ 0 & \text{otherwise} \end{cases}$$

and $Z = \sum_{x' \in \hat{\mathcal{X}}} \exp\{-U(x')\}$ yields a constrained random field formulation of the layout problem, in which configurations corresponding to a minimum cost flow have the highest probability. The random field framework for layout models was introduced in [9], and it is shown in [8] how dynamic layout models compromizing between readability and stability can be obtained through a Bayesian approach. Therefore, let Y be a random variable for the layout of $G^{(1)}$, and let X be the random variable for its successor $G^{(2)}$. The Bayesian formula gives

$$P(X = x \,|\, Y = y) = \frac{P(Y = y \,|\, X = x) \cdot P(X = x)}{P(Y = y)}$$

for all $x \in \hat{\mathcal{X}}$ and $y \in \hat{\mathcal{Y}}$. Since y is given in advance, maximization of the left hand side is equivalent to maximization of the product in the nominator of the right hand side. But the latter consists of a prior distribution $P(X = x)$ corresponding to the static layout model for $G^{(2)}$, and the likelihood of x. It was argued in [8], that this likelihood should be used to express criteria of stability. Since the only layout variables in this model are flow values directly corresponding to features of the orthogonal representation, we choose to penalize deviation in these values. In other words, we count the number of changes in angles at vertices and bends at edges. Therefore, let $P(Y = y \,|\, X = x)$ equal

$$= \begin{cases} Z_{Y|X}^{-1} \cdot \exp\left\{ -\sum_{a \in A_V^{(1)} \cap A_V^{(2)}} \alpha \cdot |x_a - y_a| - \sum_{a \in A_F^{(1)} \cap A_F^{(2)}} \beta \cdot |x_a - y_a| \right\} & \text{if } x \in \hat{\mathcal{X}}, y \in \hat{\mathcal{Y}} \\ 0 & \text{otherwise,} \end{cases}$$

where α and β are nonnegative integers, and $Z_{Y|X}$ is the normalizing constant. The dynamic model $P(X = x \,|\, Y = y)$ thus compromizes between the number of bends in an orthogonal representation (the only criterion of readability), and the number of angles changed at vertices or bends (stability). This compromise can be biased by choosing specific values for parameters α (penalty for changing an angle between consecutive edges of a face), β (penalty for introducing or removing a bend on an edge), and γ (default penalty for bends).

Summarizing the above, a reasonable objective function for layouts x of $G^{(2)}$, given a layout y of $G^{(1)}$, is

$$U(x \,|\, y) = \sum_{a \in A_F^{(2)}} \gamma \cdot x_a + \sum_{a \in A_V^{(1)} \cap A_V^{(2)}} \alpha \cdot |x_a - y_a| + \sum_{a \in A_F^{(1)} \cap A_F^{(2)}} \beta \cdot |x_a - y_a|.$$

$U(\cdot \,|\, y)$ is called the *dynamic cost function.* It is only piecewise linear. However, it is shown in the next section how minimization can still be performed by solving a minimum cost flow problem in a suitably defined network.

4 Penalized Residual Networks

Given a flow network $N = (W, A; b, l, u, c)$ and a feasible flow y, the *residual network* with respect to y, $N_y = (W, A \cup \bar{A}; b_y, l_y, u_y, c_y)$, is constructed by adding a *reduction arc* $\bar{a}$ for each $a \in A$, where $\bar{a}$ connects the same nodes as a, but is oriented in the opposite direction. $\bar{A}$ is the set of all reduction arcs. We define

– residual supplies/demands

$$b_y(w) = b(w) + \sum_{(w',w) \in A} y_{(w',w)} - \sum_{(w,w') \in A} y_{(w,w')}$$

for all $w \in W$

– residual capacities

$$\begin{aligned} l_y(a) &= l_y(\bar{a}) = 0 \\ u_y(a) &= u(a) - y_a \\ u_y(\bar{a}) &= y_a - l(a) \end{aligned}$$

– residual costs

$$c_y(a) = c(a)$$
$$c_y(\bar{a}) = -c(a)$$

Since, for now, y is a feasible flow, $b_y(w) = 0$ for all $w \in W$. Every flow z in N_y yields a circulation $\Delta(z)$ in N by setting

$$\Delta(z)_a \quad = \quad z_a - z_{\bar{a}}$$

Obviously, y can be altered by adding $\Delta(z)$ to give a new feasible flow in N. When a flow should change only if a notable improvement of the cost function is achieved, alteration can be rendered more difficult by adding penalties to residual costs. The resulting network is called the *penalized residual network*, $N'_y = (W, A \cup \bar{A}; b_y, l_y, u_y, c'_y)$, where $c'_y(a) = c_y(a) + \pi_a$ for all $a \in A \cup \bar{A}$. The nonnegative integers π_a represent the extra cost of changing flow along arc a. A flow z in the residual is called *proper*, if for all $a \in A$ at least one of z_a and $z_{\bar{a}}$ is zero. The following theorem summarizes the purpose of this construction.

Theorem 2. *Let y be a feasible flow in some network N. The cost of a proper flow z in N'_y equals $\sum_{a \in A} (c(a) \cdot (x_a - y_a) + \pi_a \cdot |x_a - y_a|)$, where $x = y + \Delta(z)$ is a feasible flow in N.*

For networks $N(G)$, parameters α and β of the dynamic cost function are used to penalize flow alteration on arcs in A_V and A_F, respectively. Figure 2 gives a small example. The following corollary of Theorem 2 shows that stable bend reduction can be carried out by computing a minimum cost flow in the penalized pseudo residual network.

Corollary 1. *Let $G^{(1)} = G^{(2)} = G$, and y be a feasible flow in $N(G)$. The cost of a proper flow z in $N'_y(G)$ equals $U(x \,|\, y) - U(y)$, where $x = y + \Delta(z)$ is a feasible flow in $N(G)$.*

The flow y in $N(G^{(1)})$ defining the orthogonal representation of $G^{(1)}$ is only a *pseudo-flow* in $N(G^{(2)})$, since its index set no longer equals the set of network arcs. In a pseudo-flow, capacity constraints are obeyed, but flow balance constraints at nodes need not. However, the definition of the penalized residual graph is easily generalized to flow vectors defined on a different index set by setting $y_a = 0$ for all a not in the index set of y. Observe that residual demands/supplies no longer equal zero in general.

Corollary 2. *Given a grid embedding of $G^{(1)}$, a grid embedding of $G^{(2)}$ according to the dynamic layout model can be computed in $\mathcal{O}((n + b)^{3/4} n \sqrt{\log n} + b)$ time, where n is the maximum number of vertices in $G^{(1)}$ and $G^{(2)}$, and b is the maximum number of bends in the orthogonal representations of $G^{(1)}$ and $G^{(2)}$.*

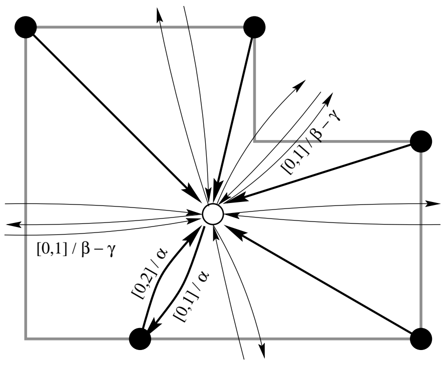


Fig. 2. Penalized residual network with respect to the flow in Fig. 1. Arcs $\bar{a} \in \bar{A}$ with capacities $l(\bar{a}) = 0 = u(\bar{a})$ are not shown. Unlabeled thick arcs have capacity $[0, 3]$ and cost α, unlabeled thin arcs have capacity $[0, \infty)$ and cost β

5 Graph Modification

In this section, we sketch one way of maintaining the penalized residual network used to evaluate the dynamic cost function for a modified graph. For convenience, we restrict updates to the *elementary* modifications of inserting and deleting an edge, where the edge may be a *bridge*, i.e. incident to a vertex of degree one. An arbitrary number of elementary modifications in the graph may be traced before a new embedding is computed. The only restriction is that the underlying graph need to be connected, embedded and planar at all times, and 4-planar in the end. Therefore, more powerful update operations can be provided by combining elementary modifications into more complex ones. For example, vertex deletion is a simple sequence of edge deletions that can be carried out in one step. Note that modifications changing the embedding of the graph require more elaborate techniques and conventions for maintaining the network.

Because of space limitations we do not go into details, but rather give an example for the construction of $N'_y(G^{(2)})$ when an edge is inserted between existing vertices of $G^{(1)}$. See Fig. 3 and note that the residual demands/supplies of the two resulting copies of the face node need to be computed by restricting the sum of flow values to those arcs that remain incident to the node. The computational demand of this modification is thus proportional to the size of the face in $G^{(1)}$. The other three elementary modifications can be performed in constant time.

6 Examples

Figure 4(a) shows a bend-minimum grid embedding of a small graph, computed according to Tamassia's model. $G^{(1)}$ is modified elementary by inserting edge $\{7, 2\}$. A bend-minimum grid embedding of the resulting graph is shown in

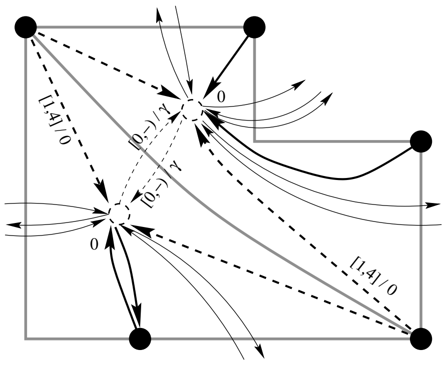


Fig. 3. Insertion of an edge. The modified network has two copies of the face node, two copies of each arc corresponding to a subdivided angle (all with suitably adjusted values), and two new arcs crossing the new edge

Fig. 4(b). Figures 4(c) and 4(d) show grid embeddings according to our dynamic model with $\alpha = \beta = \gamma = 1$, and with $\alpha = 0$, $\beta = \gamma = 1$, respectively. Changes with respect to the orthogonal representation of $G^{(1)}$ are highlighted in all three drawings of $G^{(2)}$. A vertex is rendered darker, if any angle between consecutive edges incident to it changed, and an edge is drawn thicker, if the sequence of its bends changed.

It is readily seen that the number of changes in both Fig. 4(c) and 4(d) is much smaller than in Fig. 4(b). On the other hand, the number of bends is not minimum (12 in Fig. 4(c) and 4(d) compared to 10 in Fig. 4(b)). Thus, there is an obvious compromise between the number of bends and the number of changes with respect to the orthogonal representation of Fig. 4(a).

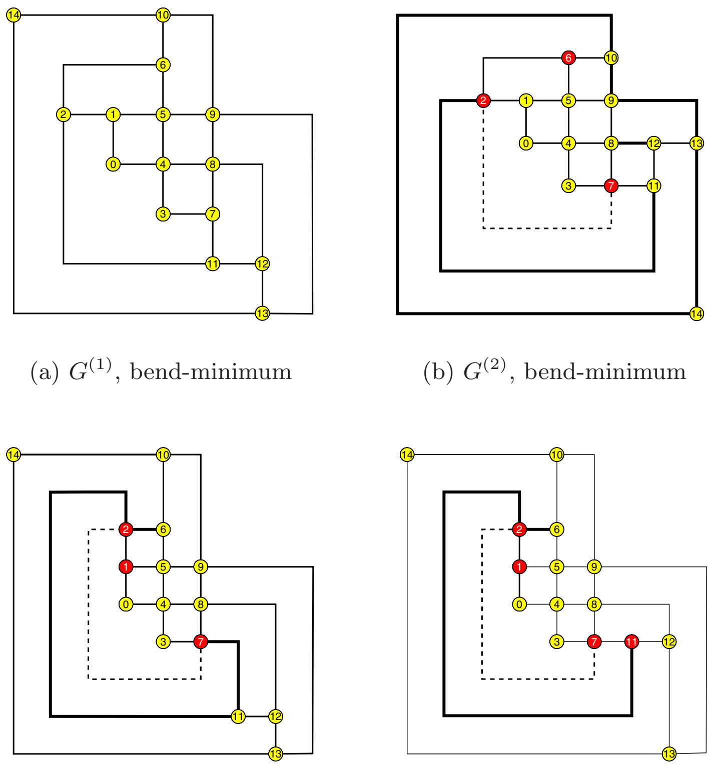
A larger example is given in Fig. 5. Again, a single edge is inserted between two existing vertices, and embeddings obtained from the bend-minimum as well as from the dynamic model are shown. By allowing one additional bend, the dynamic model clearly succeeds in maintaining the overall structure.

Acknowledgment

The authors thank Michael Güdemann for implementing the layout module used to generate the above examples.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows.* Prentice Hall, 1993. 91
2. Therese C. Biedl. New lower bounds for orthogonal graph drawings. In *Proc. Graph Drawing '95*, pages 28–39. Springer, LNCS vol. 1027, 1996. 89



(a) $G^{(1)}$, bend-minimum (b) $G^{(2)}$, bend-minimum

(c) $G^{(2)}$, dynamic (d) $G^{(2)}$, dynamic, $\alpha = 0$

Fig. 4. In both cases, the dynamic model produces an embedding with two bends more than necessary, but substantially less changes

3. Stina S. Bridgeman, Jody Fanto, Ashim Garg, Roberto Tamassia, and Luca Vismara. InteractiveGiotto: An algorithm for interactive orthogonal graph drawing. In *Proc. Graph Drawing '97*, pages 303–308. Springer, LNCS vol. 1353, 1997. 90

4. Therese C. Biedl and Goos Kant. A better heuristic for orthogonal graph drawing. In *Proc. 2nd European Symposium on Algorithms*, pages 24–33. Springer, LNCS vol. 855, 1994. 89

5. Therese C. Biedl and Michael Kaufmann. Area-efficient static and incremental graph drawings. In *Proc. 5th European Symposium on Algorithms*, pages 37–52. Springer, LNCS vol. 1284, 1997. 90

6. Therese C. Biedl, Brendan P. Madden, and Ioannis G. Tollis. The three-phase method: A unified approach to orthogonal graph drawing. In *Proc. Graph Drawing '97*, pages 391–402. Springer, LNCS vol. 1353, 1997. 89

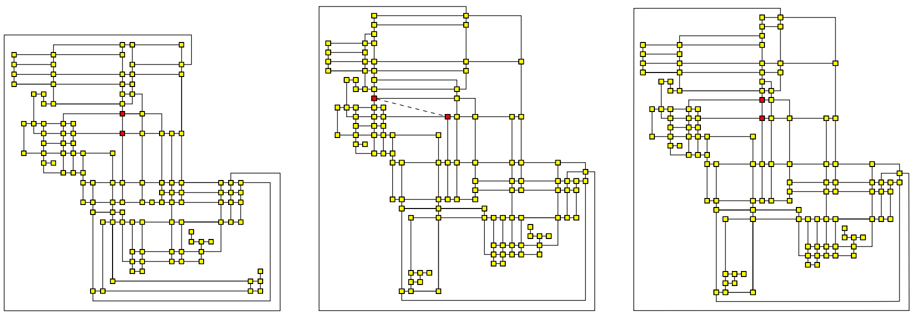


Fig. 5. A larger example with $\alpha = \beta = \gamma = 1$. The layout according to the dynamic model (*right*) displays one more bend than the minimum bend layout (*left*)

7. Karl-Friedrich Böhringer and Frances Newbery Paulisch. Using constraints to achieve stability in automatic graph layout algorithms. In *CHI'90 Proceedings*, pages 43–51. ACM, The Association for Computing Machinery, 1990. 90
8. Ulrik Brandes and Dorothea Wagner. A Bayesian paradigm for dynamic graph layout. In *Proc. Graph Drawing '97*, pages 237–247. Springer, LNCS vol. 1353, 1997. 90, 92
9. Ulrik Brandes and Dorothea Wagner. Random field models for graph layout. Konstanzer Schriften in Mathematik und Informatik 33, Universität Konstanz, 1997. 92
10. Robert F. Cohen, Giuseppe Di Battista, Roberto Tamassia, Ioannis G. Tollis, and P. Bertolazzi. A framework for dynamic graph drawing. In *Proc. 8th Symposium on Computational Geometry*, pages 261–270. ACM, The Association for Computing Machinery, 1992 90
11. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry*, 4:235–282, 1994. 89
12. Giuseppe Di Battista and Luca Vismara. Angles of planar triangular graphs. *SIAM J. Discrete Math.*, 9(3):349–359, 1996. 90
13. Peter Eades, Wei Lai, Kazuo Misue, and Kozo Sugiyama. Preserving the mental map of a diagram. *Proc. Compugraphics '91*, 9:24–33, 1991. 90
14. Ulrich Fößmeier and Michael Kaufmann. Drawing High Degree Graphs with Low Bend Numbers. In *Proc. Graph Drawing '95*, pages 254–266. Springer, LNCS vol. 1027, 1996. 90
15. Ashim Garg. On drawing angle graphs. In *Proc. Graph Drawing '94*, pages 84–95. Springer, LNCS vol. 894, 1995. 90
16. Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. In *Proc. Graph Drawing '94*, pages 286–297. Springer, LNCS vol. 894, 1995. 89
17. Ashim Garg and Roberto Tamassia. An new minimum cost flow algorithm with applications to graph drawing. In *Proc. Graph Drawing '96*, pages 201–216. Springer, LNCS vol. 1190, 1997.

18. Seth Malitz and Achilleas Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.*, 7(2):172–183, 1994. 90
19. Stephen North. Incremental layout in DynaDAG. In *Proc. Graph Drawing '95*, pages 409–418. Springer, LNCS vol. 1027, 1996. 90
20. Achilleas Papakostas and Ioannis G. Tollis. Improved algorithms and bounds for orthogonal drawings. In *Proc. Graph Drawing '94*, pages 40–51. Springer, LNCS vol. 894, 1995. 89
21. Achilleas Papakostas and Ioannis G. Tollis. Issues in interactive orthogonal graph drawing. In *Proc. Graph Drawing '95*, pages 419–430. Springer, LNCS vol. 1027, 1996. 90
22. Helen Purchase. Which aesthetic has the greatest effect on human understanding? In *Proc. Graph Drawing '97*, pages 248–261. Springer, LNCS vol. 1353, 1997. 90
23. Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. 89, 90, 91
24. Roberto Tamassia, Giuseppe DiBattista, and Carlo Batini. Automatic graph drawing and readabilityof diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988. 90
25. Gopalakrishnan Vijayan. Geometry of planar graphs with angles. In *Proc. 2nd Symposium on Computational Geometry*, pages 116–124. ACM, The Association of Computing Machinery, 1986. 90

Two New Families of List Update Algorithms

Frank Schulz[⋆]

Universität des Saarlandes, FB 14, Lehrstuhl Prof. G. Hotz
Postfach 151150, 66041 Saarbrücken, Germany
schulz@cs.uni-sb.de

Abstract. We consider the online list accessing problem and present a new family of competitive-optimal deterministic list update algorithms which is the largest class of such algorithms known to-date. This family, called Sort-by-Rank (sbr), is parametrized with a real $0 \leq \alpha \leq 1$, where sbr(0) is the Move-to-Front algorithm and sbr(1) is equivalent to the Timestamp algorithm. The behaviour of sbr(α) mediates between the eager strategy of Move-to-Front and the more conservative behaviour of Timestamp.

We also present a family of algorithms Sort-by-Delay (sbd) which is parametrized by the positive integers, where sbd(1) is Move-to-Front and sbd(2) is equivalent to Timestamp. In general, sbd(k) is k-competitive for $k \geq 2$. This is the first class of algorithms that is asymptotically optimal for independent, identically distributed requests while each algorithm is constant-competitive.

Empirical studies with with both generated and real-world data are also included.

1 Introduction

Consider a set $\{1, \ldots, n\}$ of n items that are to be stored as an unsorted linear list. A request for an item x is processed by searching the list for x starting from the head of the list. The cost of the retrieval is assumed to be the number of comparisons necessary to identify the item, i.e. the number of the position at which the item was found. After the retrieval, a list update algorithm may move the requested element to any position closer to the front without additional costs (*free exchanges*). Any other swap of two consecutive items incurs cost 1 (*paid exchanges*). The goal of such exchanges is to minimize the cost for future accesses. When the list update algorithm processes the requests without knowing future queries, it is said to work *online*.

In the sequel we will only consider requests to items and no insertions or deletions (static model). It is possible to extend the analyses to the dynamic model where the set of list items may change over time, the results carry over to this situation.

⋆ supported by a Graduiertenkolleg fellowship from the Deutsche Forschungsgemeinschaft (DFG), Germany.

List update algorithms are interesting because they are frequently used in practice. The most direct application is the implementation of a dictionary as a singly-linked list. As online algorithms constantly try to adapt to the request sequence, they can approximate the process that generated the requests. Moreover, this is achieved in a computationally inexpensive way. As such, list update schemes have been successfully applied in data compression algorithms [1,5] Surprisingly, online algorithms also gain importance in geometric applications such as computing point maxima or convex hulls [9].

In competitive analysis, an online algorithm ALG is compared to an optimal offline algorithm OPT. Let σ be any sequence of requests to items in the list (we assume that all queries are successful, i.e. every requested item is actually in the list). Denote by ALG(σ) the total cost incurred by ALG to serve σ, and OPT(σ) the total cost incurred by an optimal offline algorithm. Then ALG is said to be c-competitive (having a competitive ratio of c) if there exists a constant a such that $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + a$ for all request sequences σ.

In average case analysis, the request sequence is generated according to some probability model. In the independent model, a probability distributions over the list items is given. The requests are the realization of a sequence of independent, identically distributed random variables with this distribution (the requests are generated by some discrete memoryless source). In the markov model, the requests are generated by a (first order) Markov chain. In those models, the asymptotic expected costs of online algorithms and of optimal offline algorithms and their ratios are of main interest.

1.1 On Previous Work

The most famous list update algorithm is Move-to-Front, it has been studied since 1965 [12].

> Move-to-Front (MTF): Upon a request for item x, move x to the front of the list.

In their seminal work on competitive analysis [14], Sleator and Tarjan showed that MTF ist 2-competitive. Raghavan and Karp proved a lower bound of $2 - 2/(n+1)$ on the competitive ratio of any deterministic list update scheme and list size n. Hence, MTF is a competitive-optimal algorithm[1].

The important Timestamp algorithm discovered recently by Albers [2] is 2-competitive as well.

> Timestamp (TS): Upon a request for item x, insert x in front of the first item y that precedes x and that was requested at most once since the last request for x. Do nothing if there is no such item y, or if x has been requested for the first time.

[1] All algorithms with competitive ratio 2 will be called optimal. As MTF does indeed have a competitive ratio of $2 - 2/(n+1)$, it is sometimes called *strongly-optimal*.

Both algorithms have optimal competitive ratio, nevertheless they are rather different. This difference is blurred by competitive analysis, but it is revealed when average case analysis is employed. In the independent model, the ratio between the expected cost of the MTF algorithm and a static optimal algorithm can be arbtrarily close to $\pi/2 = 1.5707\ldots$ [10]. For TS, this ratio is less than 1.34 on any access distribution [1]. On the other hand, for Markov sources with the phenomenon of locality, MTF fares better. This demonstrates that TIMESTAMP reacts more conservatively, and MTF more eagerly.

Recently, a new family of list update algorithms has been introduced by El-Yaniv [8]. The MOVE-TO-RECENT-ITEM(k) algorithms are located "between" the MTF and TS strategies.

> MOVE-TO-RECENT-ITEM(k), MRI(k): Upon a request for item x, move x forward just after the last item y that is in front of x and that was requested at least $k+1$ times since the last request for x. If there is no such item y of if this is the first request for x, move x to the front.

It has been shown that MRI(1) is equivalent to the TIMESTAMP algorithm, and that MOVE-TO-FRONT is the limit element of that family, as $k \to \infty$. Moreover, for all $k \geq 1$ MRI(k) is 2-competitive. Hence there is an infinite (countable) class of competitive-optimal list update algorithms.

1.2 New Results

We define a new infinite (uncountable) family of list update schemes that turn out to be competitive-optimal. Let $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$ be any request sequence. Consider any point t, $1 \leq t \leq m$, in time, and define for each list item x the last access time $w_1(x,t)$,

$$w_1(x,t) = \max(\{t' \leq t : \sigma_{t'} = x\} \cup \{0\}) .$$

For $k > 1$ define the kth last access time $w_k(x,t)$ to x at time t,

$$w_k(x,t) = \max(\{t' < w_{k-1}(x,t) : \sigma_{t'} = x\} \cup \{0\}) .$$

Now consider for $k \geq 1$ the rank functions

$$s_k(x,t) = t - w_k(x,t) .$$

Lemma 1. *(a) For all t, such that each list item has been requested at least once, we have: When the list is maintained by* MOVE-TO-FRONT*, the current list ordering is $[x_1, \ldots, x_n]$ iff*

$$s_1(x_1,t) < s_1(x_2,t) < \cdots < s_1(x_n,t).$$

(b) For all t, such that each list item has been requested at least twice, we have: When the list is maintained by TIMESTAMP*, the current list ordering is $[x_1, \ldots, x_n]$ iff*

$$s_2(x_1,t) < s_2(x_2,t) < \cdots < s_2(x_n,t).$$

Due to lack of space, the proof is omitted. Note that this approach gives a very natural interpretation of the Timestamp rule.

For $\alpha \geq 0$, define a new rank function as the convex combination of $s_1(x,t)$ and $s_2(x,t)$:

$$r_t(x) = r_t(x,\alpha) \; = \; (1-\alpha)\cdot s_1(x,t) + \alpha \cdot s_2(x,t).$$

Then we define the new family of list update algorithms.

> Sort-by-Rank(α), sbr(α): Upon a request for item x at time t, insert x just after the last item y in front of x with $r_t(y,t) < r_t(x,t)$. Insert x in front of the list if there is no such item y, or when x is requested for the first time.

Note that sbr(0) = mtf and sbr(1) is equivalent to ts modulo the handling of first requests. This means that sbr(1) and ts acted similarly if ts would be changed such that it moves an item to the front of the list upon its first request.

Theorem 1. *For $0 \leq \alpha \leq 1$, the algorithm* sbr(α) *is 2-competitive.*

This result provides the first example of an uncountable class of competitive-optimal list update algorithms. The sbr(α) schemes use $2n$ timestamps for all α. In contrast, the mri(k) rules need nk timestamps which is significant when considering rules “close” to mtf. We remark that in general, the sbr(α) algorithms do not share the *pairwise independence property* that is satisfied by mtf and ts. Hence the proof is based on list factoring and a potential function argument similar to the one used in [8]. Note that for $0 \leq \alpha \leq 1$, the sbd(α) schemes interpolate between mtf and ts. For $\alpha > 1$, sbd(α) “extrapolates”, and in the limit $\alpha \to \infty$, the list is sorted by increasing time *differences* $s_2(x,t) - s_1(x,t)$ between the second last and the last access to x. However, in these cases, the algorithms are no longer 2-competitive, as can be easily shown by example.

The second class of list update schemes that we examine is

> Sort-by-Delay(k), sbd(k): Upon a request for item x at time t, insert x just after the last item y in front of x with $s_k(y,t) < s_k(x,t)$. Insert x in front of the list if there is no such item y, or when x is requested for the first time.

Here we have sbd(1) = mtf and sbd(2) is equivalent to ts in the above sense. Thus, sbd(k) turns out to be a natural generalization of mtf and ts.

Theorem 2. *For every $k \geq 2$,* sbd(k) *is k-competitive.*

Lemma 2. *An equivalent definition of* sbd(k) *is the following:*

> *Upon a request for item x, insert x in front of the first item y that precedes x and that was requested at most $k-1$ times since the $(k-1)$th last access to x. Insert x in front of the list if there is no such item y, or if x has been requested for the first time.*

It turns out that the SBD(k) rules do satisfy the pairwise independence property. This makes it easy to achieve an average case analysis.

Theorem 3. *For any probability distribution $p = (p_1, \ldots, p_n)$, the asymptotic expected costs per request for* SBD*(k) are*

$$E_{\mathrm{SBD}(k)}(p) = \sum_{1 \le i \le j \le n} \sum_{\ell=0}^{k-1} \binom{2k-1}{\ell} \frac{p_i^{\ell+1} p_j^{2k-1-\ell} + p_i^{2k-1-\ell} p_j^{\ell+1}}{(p_i + p_j)^{2k-1}} .$$

For $k = 1$, this has been found in [12], and for $k = 2$ see [1]. We compare this expected cost with the optimal cost $M(p)$ incurred by an optimal static adversary. For $p_1 \ge p_2 \ge \cdots \ge p_n$, we have $M(p) = \sum_{i=1}^{n} i p_i$.

Theorem 4. *The family* SBD*(k) is asymptotically optimal, i.e. for all distributions p we have*

$$E_{\mathrm{SBD}(k)}(p) \longrightarrow M(p) \qquad \text{for } k \to \infty.$$

In particular, the cost ratios are bounded as follows

$$\begin{aligned} E_{\mathrm{SBD}(1)}(p)/M(p) &\le 1.5708 \\ E_{\mathrm{SBD}(2)}(p)/M(p) &\le 1.3391 \\ E_{\mathrm{SBD}(3)}(p)/M(p) &\le 1.3044 \\ E_{\mathrm{SBD}(4)}(p)/M(p) &\le 1.2904 \\ E_{\mathrm{SBD}(5)}(p)/M(p) &\le 1.2828. \end{aligned}$$

For $k = 1$ (MTF), the ratio is bounded by $\pi/2 \le 1.5708$, as proved in [7], and for $k = 2$ (TS), the ratio has been bounded in [1].

While the SBD(k) algorithms are not competitive-optimal, they are constant-competitive and, under the independent model, achieve an expected search time that is arbirtrarily close to the optimal expected search time of an static adversary. To the best of our knowledge, all previous list update algorithms with this property, e.g. FREQUENCY-COUNT, SIMPLE(k)-MTF or BATCHED(k)-MTF have a competitive ratio that is linear in the length n of the list, for all $k \ge 2$.

The constant-competitiveness makes the SBD(k) schemes attractive for use with request sequences with a low degree of locality. In particular, the empirical studies suggest that there is a parameter k^* that suits best to the degree of locality of a given sequence, i.e. SBD(k^*) achieves the lowest cumulative access costs among all SBD(k) schemes.

In *section 2*, the Sort-by-Rank algorithms are investigated and theorem 1 is proved. *Section 3* outlines the analysis of the Sort-by-Delay rules. In *section 4*, empirical results for both families are reported. They were obtained from streams of independent identically distributed requests and from request sequences obtained from the Calgary Compression Corpus [15], a standard benchmark collection for file compression. *Section 5* concludes with some remarks and open questions.

2 Analysis of the Sort-by-Rank Algorithms

First we give an example showing that the SBR(α) rules do not in general satisfy the pairwise independence property. This property means that for every request sequence σ and two arbitrary items x and y in the list L, their relative order is the same when σ is served as their relative order in L_{xy} (the two-element list holding x and y) when the projected sequence σ_{xy} is served [4]. The projeted sequence σ_{xy} is defined as σ after deletion of all requests for items other than x and y. Consider the request sequences $\sigma_1 = x, y, y, a, x$ and $\sigma_2 = x, a, y, y, x$. The projection onto $\{x, y\}$ gives in both cases $\sigma_{xy} = x, y, y, x$. But for $\alpha = 1/2$ and $t = 5$, we have in the first case $r_t(x) = 2 < 2.5 = r_t(y)$ and in the second case $r_t(x) = 2 > 1.5 = r_t(y)$.

For the proof of theorem 1, we need some preliminaries on list factoring. This technique provides lower bounds on the cost OPT(σ) of the optimal algorithm on the request sequence σ by considering a collection of optimally maintained two-element lists. Let FOPT be an optimal "factored" offline algorithm that maintains the collection of all $\binom{n}{2}$ two-element lists L_{xy} for all sets $\{x, y\} \subseteq L$. Hence, each such list L_{xy} is controlled by an optimal offline algorithm that serves the request sequence σ_{xy}. Denote by FOPT_t the cost of serving σ_t which is incurred by FOPT, i.e. the number of two-element lists with σ_t in the second position. Then, for any request sequence $\sigma = \sigma_1, \ldots, \sigma_m$, we have the following lower bound for the optimal offline cost:

$$\sum_{t=1}^{m} \text{FOPT}_t \leq \text{OPT}(\sigma). \tag{1}$$

Further details are omitted due to the lack of space, they may be found in the full version or in [8].

Let L be the list maintained by SBR(α). When item x is currently in front of item y in L, We say $\langle x, y\rangle$ in L. When in the same situation y is in front of x in the list L_{xy} maintained by FOPT, then $\langle x, y\rangle$ is called an inversion.

For each pair of items $x \neq y$, we define the weight function

$$w(x,y) = \left\{ \begin{array}{ll} 0: \text{if } \langle x,y\rangle & \text{is not an inversion} \\ 1: \text{if } \langle x,y\rangle & \text{is an inversion and } y \text{ will pass } x \\ & \text{in } L \text{ if } y \text{ is requested next} \\ 2: \text{if } \langle x,y\rangle & \text{is an inversion and } y \text{ will pass } x \\ & \text{in } L \text{ iff } y \text{ is now requested twice in a row} \end{array} \right\}.$$

Then we define the potential function $\Phi = \sum_{x \neq y} w(x, y)$.

The potential is always non-negative. Fix a request sequence $\sigma = \sigma_1, \ldots, \sigma_m$. The *amortized cost* a_t for SBR(α) to serve σ_t is defined as $a_t = \text{SBR}(\alpha)_t + \Phi_t - \Phi_{t-1}$, where $\text{SBR}(\alpha)_t$ is the actual cost incurred by SBR(α) in this step, and Φ_t is the potential at time t. The proof demonstrates that there is a constant $c = 2$ such that $\text{SBR}(\alpha)_t \leq c \cdot \text{FOPT}_t$ for all requests $t = 1, \ldots, m$. Then it follows from equation (1) that SBR(α) is 2-competitive.

Lemma 3. *Let y and z be two items such that $\langle z, y\rangle$ in L is an inversion with weight $w(z, y) = 1$. Let x be any item other than y or z. Then $w(z, y)$ remains 1 after a request for x is served by* SBR(α).

Let the current request be for item $\sigma_t = x$. Without loss of generality we assume that each element has been requested at least once. Define the set $\mathcal{A}$ as the collection of all items z such that $\langle z, x\rangle$ in L is not an inversion; the set $\mathcal{B}$ as the set of all z such that $\langle z, x\rangle$ in L is an inversion, and the set $\mathcal{C}$ as the collection of all elements z such that $\langle x, z\rangle$ in L is an inversion. Note that these three sets are pairwise disjoint. Set $A = |\mathcal{A}|$, $B = |\mathcal{B}|$ and $C = |\mathcal{C}|$.

Let the actual cost at time t be SBR$(\alpha)_t$. This is equal to $A + B + 1$, as the elements in front of x are exactly the items from $\mathcal{A} \cup \mathcal{B}$. The cost incurred in the optimally maintained factored list is FOPT$_t = A + C + 1$, because exactly the sets $\{z, x\}$ with $z \in \mathcal{A} \cup \mathcal{C}$ form a contribution to this cost.

Now we consider the change in potential due to the action of SBR(α) when serving $x = \sigma_t$. If an element $z \in \mathcal{B}$ is passed by x, the weight of the inversion $\langle z, x\rangle$ in L was 1 before the move, and it is now eliminated. If an element $z \in \mathcal{B}$ is not passed by x, the weight of the inversion $\langle z, x\rangle$ in L was 2 before the move, and is now reduced to 1. Hence there is a decrease in potential of exactly B due to items in $\mathcal{B}$. On the other hand, due to SBR(α)'s move, the weight of each inversion $\langle x, z\rangle$ with $z \in \mathcal{C}$ may increase by one which results in a total increase of at most C.

Next we examine the change in potential due to the action of the optimal offline algorithm FOPT maintaining the factored list when serving $x = \sigma_t$. Suppose that a new inversion $\langle x, z\rangle$ in L is created due to the fact that x has passed z in L, but not in L_{xz}. Then $z \in \mathcal{A}$ before the request and we use the following lemma:

Lemma 4. *Let $0 \le \alpha \le 1$. If, at time t in the list L, item $\sigma_t = x$ has passed an element $z \in \mathcal{A}$, then $s_1(z, t) \le s_2(x, t)$. If this request created an inversion $\langle x, z\rangle$ in L, and the next request is for $\sigma_{t+1} = z$, then z will pass x. In other words, this inversion has weight $w(x, z) = 1$.*

Next suppose that a new inversion $\langle x, z\rangle$ in L is created due to the fact that x has passed z in L_{xz}, but not in L. Then $z \in \mathcal{A}$ before the request, and the weight of the inversion can only be one, since a subsequent request for $x = \sigma_{t+1}$ would move x to the front and remove this inversion. Hence, the weight of the new inversions is at most A.

In total, the change in potential after serving the request σ_t can be bounded as $\Phi_t - \Phi_{t-1} \le C - B + A$. The amortized costs for SBR(α) to serve σ_t are

$$\begin{aligned} a_t &= \text{SBR}(\alpha)_t \ + \ \Phi_t - \Phi_{t-1} \\ &\le A + B + 1 \ + \ C - B + A \\ &= (2 - (C+1)/(A+C+1)) \cdot (A + C + 1) \\ &\le (2 - 1/n) \cdot \text{FOPT}_t. \end{aligned}$$

This completes the proof that SBR(α) is 2-competitive for $0 \le \alpha \le 1$.

3 Analysis of the Sort-by-Delay Algorithms

The proof of SBD(k) being k-competitive for $k \geq 2$ is a modification of the proof of theorem 1 with a different potential function; it is based on the following weight function

$$w(x,y) = \left\{ \begin{array}{l} 0 : \text{if } \langle x,y \rangle \text{ is not an inversion} \\ \ell : \text{if } \langle x,y \rangle \text{ is an inversion and } y \text{ will pass } x \text{ in } L \\ \qquad\qquad\quad \text{iff } y \text{ is now requested } \ell \text{ times, } \ell = 1, \ldots, k \end{array} \right\} .$$

The pairwise independence property of the SBD(k) rules is an immediate consequence of lemma 5.

Lemma 5. *Upon a request for item x, all items that were requested at least k times since the $(k-1)$th last request for x are in front of all items that were requested less than k times since the $(k-1)$th last request for x.*

Lemma 6. *Consider any point in the request sequence where there have been at least $2k-1$ requests to x or y. Then y precedes x if and only if the majority (i.e. at least k) of the $2k-1$ last requests to x or y have been for y.*

The proof of theorem 3 is based on lemma 6. Similarly, lemma 6 can be used to obtain explicit formulæ for the expected search cost of SBD(k) for Markov dependent request sequences. The first part of theorem 4 follows with a Chernoff bound argument, in the second part the calculations are similar to [7].

4 Empirical Results

In figure 1, the simulation results of SBR(α) for the Zipf distribution ($p_i \propto 1/i$) are presented for $0 \leq \alpha \leq 1$, and the average search cost on a list with $n = 50$ items is depicted. The SBR(α) rules provide a gradual transition from MTF to TS. This confirms our intuition in the average case setting where analytical results are difficult to achieve.

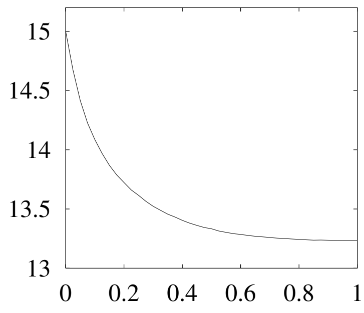


figure 1

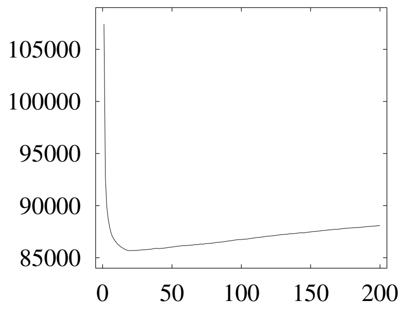


figure 2

We conducted compression experiments with the SBD(k) rules on the Calgary/ Canterbury Compression Corpus [15]. The list update schemes work as "engines" that dynamically define a statistical model for the data. Initially the list is sorted by decreasing access frequencies of the characters. Then the characters are encoded sequentially. When a character has been found on position i of the list, its position is encoded with Elias encoding, a standard variable length prefix-free code for positive integers, using $1 + 2\lfloor \log_2(i) \rfloor$ bits.

For example, the compression performance of SBD(k) on the file *geo* is depicted in figure 2 for $1 \leq k \leq 200$. The value k^* shall describe the rule with the best compression among all SBD(k) schemes. Hence the adaptive discipline SBD(k^*) fits best to the particular degree of locality found in the input stream. Sometimes however, the value k^* is not sharply distinguished. Therefore, k^* can not give a precise quantification of locality in general.

In the following table, the results are collected and compared with the sizes of the original file and the results achieved by MOVE-TO-FRONT compression (MTF) and by FREQUENCY COUNT compression (FC). The latter uses as statistical model a static list that is ordered by decreasing access frequencies. The results of SBD(k^*) are significantly better than MTF compression, with typical savings around 15 %. However, in comparison with FREQUENCY COUNT, SBD(k^*) hardly manages to break even. The good performance of FC on the Calgary Compression Corpus has already been observed in [3].

File	Original	FC	MTF	k^*	SORT-BY-DELAY(k^*)			
	Bytes	Bytes	Bytes		Bytes	% Orig	% FC	% MTF
bib	111261	88641	106468	73	89145	80.1	100.6	83.7
book1	768771	516198	644418	153	516966	67.2	100.1	80.2
book2	610856	434798	515255	23	432940	70.9	99.6	84.0
geo	102400	85533	107422	19	85687	83.7	100.2	79.8
news	377109	292800	333729	18	289925	76.9	99.0	86.9
obj1	21504	19304	19351	3	17802	82.8	92.2	92.0
obj2	246814	237165	250952	7	219668	89.0	92.6	87.5
paper1	52161	39617	46140	25	39066	73.5	98.6	84.7
paper2	82199	56356	69437	54	56539	68.8	100.3	81.4
paper3	46526	32497	39373	35	32697	70.3	100.6	83.0
paper4	13286	9229	11273	30	9327	70.2	101.1	82.7
paper5	11954	8741	10195	18	8759	73.3	100.2	85.9
paper6	38105	28487	32073	13	27320	71.7	95.9	85.2
pic	513216	111607	119125	7	109733	21.4	98.3	92.1
progc	39611	30779	39150	13	30414	76.8	98.8	77.7
progl	71646	51260	55178	14	48776	68.1	95.2	88.4
progp	49379	35066	40041	9	34903	70.7	99.5	87.2
trans	93695	82297	82055	5	76895	82.1	93.4	93.7

We are optimistic that the possibility to adapt to the degree of locality is fruitful in other applications. Similarly, the SBR(α) schemes could be used for fine-tuning to data streams with very high level of locality, such as the ouput of the Burrows-Wheeler-transform [6].

5 Conclusion

We introduced and analysed two families of deterministic list update algorithms, called Sort-by-Rank(α) and Sort-by-Delay(k). Empirical studies with sequences from the Calgary Compression Corpus suggest that the parameter k^* which denotes the rule SBD(k^*) with the best compression performance could be considered as a crude measure of the degree of locality of that sequence. The question of quantifying the locality of reference in data streams remains a challenge of major importance.

It would also be interesting to experiment with non-linear rank functions, for example $r_t(x) = (s_1(x,t)^p + s_2(x,t)^p)^{1/p}$. Finally, an open question is to improve the tradeoff between competitive ratio and expected search cost on memoryless sources as exhibited by the SBD(k) family, or to prove a lower bound.

Acknowledgements

The discussions with Günter Hotz and the comments by Susanne Albers are greatly appreciated.

References

1. S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms. In *ICALP'96*, LNCS 1099, pages 514–525. Springer, 1996. 100, 101, 103
2. S. Albers. Improved randomized on-line algorithms for the list update problem. In *SODA'95*, pages 412–419, 1995. 100
3. Ran Bachrach and Ran El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *SODA'97*, pages 53–62, 1997. 107
4. J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985. 104
5. J. L. Bentley, D. Sleator, R. E. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29:320–330, 1986. 100
6. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report 124, DEC Systems Research Center, May 1994. 108
7. F. R. K. Chung, D. J. Hajela, and P. D. Seymour. Self-organizing sequential search and Hilbert's inequalities. In *Proc. 17th ACM STOC*, pages 217–223, 1985. 103, 106
8. Ran El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Manuscript, 1997. 101, 102, 104
9. M. J. Golin. A provably fast linear-expected-time maxima-finding algorithm. *Algorithmica*, 11:501–524, 1994. 100

10. G. H. Gonnet, J. I. Munro, and H. Suwanda. Exegesis of self-organizing linear search. *SIAM Journal of Computing*, 10(3):613–637, 1981. 101
11. K. Lam, M. Y. Leung, and M. K. Siu. Self-organizing files with dependent accesses. *Journal of Applied Probability*, 21:343–359, 1984.
12. J. McCabe. On serial files with relocatable records. *Oper. Res.*, 12:609–618, 1965. 100, 103
13. F. Schulz and E. Schömer. Self-organizing data structures with dependent accesses. In *ICALP'96*, LNCS 1099, pages 526–537. Springer, 1996.
14. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. 100
15. I. H. Witten and T. Bell. The calgary text compression corpus. Anonymous ftp from `ftp.cpsc.ucalgary.ca:/pub/projects/text.compression.corpus/`. 103, 107

An Optimal Algorithm for On-Line Palletizing at Delivery Industry
(Extended Abstract)

J. Rethmann and E. Wanke

Dept. of Computer Science, University of Düsseldorf
D-40225 Düsseldorf, Germany
{rethmann,wanke}@cs.uni-duesseldorf.de

Abstract. We consider the combinatorial stack-up problem which is to process a given sequence q of bins by removing step by step bins from the first s positions of the sequence onto p stack-up places. We prove that the Most-Frequently algorithm has best worst-case performance of all *on-line* stack-up algorithms and is, additionally, the best polynomial time approximation algorithm for the stack-up problem known up to now, although it is a simple *on-line* algorithm.

1 Introduction

In this paper, we consider the combinatorial problem of stacking up bins from a conveyer onto pallets. This problem basically appears in so-called *stack-up systems* which play an important role in delivery industry and warehouses. Customers order a large amount of articles, which have to be packed into bins for delivery. This work is usually done in so-called *pick-to-belt orderpicking systems*, see [dK94,LLKS93]. Typically, not all articles ordered by one customer fit into a single bin, so each order is divided into several bins. For delivery, it is necessary that all bins belonging to one order are placed onto the same pallet, and bins for different orders are placed onto different pallets. Bins arrive the stack-up system on the main conveyer of the orderpicking system. At the end of the main conveyer they enter a *storage conveyer.* The bins are picked-up by stacker cranes from the storage conveyer and moved onto pallets, which are located at so-called *stack-up places.* Automatic driven vehicles take full pallets from stack-up places, put them onto trucks, and bring new empty pallets to the stack-up places, see also [RW97c].

Logistic experience over 10 years lead to such high flexible conveyer systems in delivery industry. So we do not intend to modify the architecture of existing systems, but try to develop a model that captures important parameters of stack-up systems, and to develop efficient and provable good algorithms to control them. Today, real stack-up systems are controlled by fuzzy logic. Since almost all decisions have to be done at a time where only a part of the complete sequence is known, we focus our work on on-line algorithms.

From a theoretical point of view, we are given a sequence of pairwise distinct bins $q = (b_1, \dots, b_n)$ and two positive integers s and p. Each bin b_i of q is destined for one pallet, where the number of bins for a pallet is arbitrary. The bins have to be removed step by step from q. If a bin is removed from the sequence, then all bins to the right are shifted one position to the left. The position of a removed bin must not be greater than s, and after each removal of a bin there must be at most p pallets open. A pallet t is called open, if at least one bin for t is already removed from sequence q, but at least one bin for t is still contained in the current sequence. Integer s corresponds to the capacity of the storage conveyer, and integer p corresponds to the number of stack-up places, because each open pallet occupies one stack-up place. A sequence q is called an (s, p)-sequence if it can be processed with a storage capacity of s bins and p stack-up places.

The following results are already known about the stack-up problem. In [RW97c] it is shown that the stack-up decision problem is NP-complete [GJ79], but belongs to NL if the storage capacity s or the number of stack-up places p is fixed. In [RW97b] the performances of several *on-line* algorithms are compared with optimal off-line solutions by a competitive analysis [MMS88]. The Most-Frequently algorithm – or MF algorithm for short – processes each (s, p)-sequence with a storage capacity of $(p + 1)s - p$ bins and p stack-up places, or with a storage capacity of s bins and at most $p \cdot (\log_2(s) + 2)$ stack-up places, respectively. In [RW97a], a polynomial time *off-line* approximation algorithm for the processing of (s, p)-sequences with a storage capacity of $s \cdot \lceil \log_2(p + 1) \rceil$ and $p + 1$ stack-up places is introduced.

In contrast to the performance analysis given in [RW97b], we show in this paper that the MF algorithm is even an *optimal* on-line stack-up algorithm. Furthermore, we do not only measure the performance in terms of one objective, but we also relax both resources, the storage capacity as well as the number of stack-up places. We look at on-line stack-up algorithms that take a storage capacity of $s + c$ bins to process (s, p)-sequences, where c is any nonnegative integer. We give a lower bound on the number of stack-up places each such algorithm takes, and prove an upper bound of $p + p \cdot \log_2(\frac{s}{c+1} + 1) + 1$ the MF algorithm takes. Furthermore, we show that the MF algorithm processes each (s, p)-sequence q with a storage capacity of $2s$ bins and $2p$ stack-up places. Therefore, the MF algorithm is the best polynomial time approximation algorithm for the stack-up problem known up to now that approximates both objectives within a small factor. We emphasize that this approximation can be achieved by a very simple *on-line* algorithm. So Most-Frequently seems to be the right algorithm to put into practice.

2 Preliminaries

Throughout the paper s and p are positive integers, and c is a nonnegative integer. We consider *sequences* $q = (b_1, \dots, b_n)$ of n pairwise distinct *bins*. A sequence $q' = (b_{i_1}, \dots, b_{i_k})$ is a *subsequence* of q if $1 \le i_1 < i_2 < \dots < i_{k-1} < i_k \le n$. The *position* j of a bin b_{i_j} in sequence $q' = (b_{i_1}, \dots, b_{i_k})$ is denoted by

$pos(q', b_{i_j})$. If X is the set of bins removed from q to obtain q' then q' is also denoted by $q - X$.

Each bin b of q is associated with a *pallet symbol* $plt(b)$ which is in general some positive integer. We say bin b is destined for pallet $plt(b)$. The labeling of the pallets is insignificant, because we only need to know whether or not two bins are destined for the same pallet. The set of all pallets of the bins in sequence q is denoted by $plts(q) := \{ \ plt(b) \mid b \in q \ \}$.

Suppose we remove step by step some bins from a sequence $q = (b_1, \ldots, b_n)$. The successive removal of k bins yields k subsequences

$$q_0 = q, \ q_1 = q - \{b_{j_1}\}, \ q_2 = q - \{b_{j_1}, b_{j_2}\}, \ \ldots, \ q_k = q - \{b_{j_1}, \ldots, b_{j_k}\}$$

of q. Each pair (q, q_i) is called *configuration.* A pallet t is defined to be *open* in q_i if and only if some bin for pallet t is already removed from q, but sequence q_i still contains at least one bin for pallet t. The set of all open pallets in configuration (q, q_i) is denoted by $open((q, q_i))$.

The removal of some bin b from subsequence q_i is called (s,p)*-transformation step*, if

$$pos(q_i, b) \leq s, \qquad |open((q, q_i))| \leq p \qquad \text{and} \qquad |open((q, q_i - \{b\}))| \leq p.$$

A sequence of (s,p)-transformation steps that transforms sequence q into some subsequence q_i of q is called (s,p)*-transformation* of q into q_i. The integers s and p represent the used *storage capacity* and the used number of *stack-up places.* An (s,p)-transformation of q into some empty sequence is called (s,p)*-processing* of q. If such an (s,p)-processing exists for sequence q, then q is called an (s,p)-*sequence.* The *stack-up problem* is to decide whether there is an (s,p)-processing for a given sequence q and given positive integers s and p.

It is sometimes convenient to use pallet identifications instead of bin identifications. So we will write $q \mathrel{\hat{=}} (plt(b_1), \ldots, plt(b_n))$ instead of $q = (b_1, \ldots, b_n)$.

Example 1. Consider sequence $q \mathrel{\hat{=}} (1,1,2,3,2,3,2,3,2,1)$. The table below shows a $(3,2)$-processing of sequence q. The underlined bin is the bin which will be removed in the next transformation step. The columns marked S_i and j_i will be explained later.

i	$q_i \mathrel{\hat{=}}$	$open((q, q_i))$	S_i	j_i
0	$(1,1,\underline{2},3,2,3,2,3,2,1)$	$\emptyset$	$1,1,2$	4
1	$(1,1,\underline{3},2,3,2,3,2,1)$	$\{2\}$	$1,1,3$	5
2	$(1,1,\underline{2},3,2,3,2,1)$	$\{2,3\}$	$1,1,2$	6
3	$(1,1,\underline{3},2,3,2,1)$	$\{2,3\}$	$1,1,3$	7
4	$(1,1,\underline{2},3,2,1)$	$\{2,3\}$	$1,1,2$	8
5	$(1,1,\underline{3},2,1)$	$\{2,3\}$	$1,1,3$	9
6	$(1,1,\underline{2},1)$	$\{2\}$	$1,1,2$	10
7	$(\underline{1},1,1)$	$\emptyset$	$1,1,1$	11
8	$(\underline{1},1)$	$\{1\}$	$1,1$	11
9	$(\underline{1})$	$\{1\}$	1	11
10	$()$	$\emptyset$		11

Consider an (s, p)-transformation of sequence q into some sequence q'. Let S denote the sequence consisting of the first s bins of q', and let b_j be the first bin of the last $|q'| - s$ bins of q'. Then we can uniquely specify configuration (q, q') by some triple (q, S, j). Set S is called the *storage*. If q' contains less than or equal to s bins then j is defined by $|q| + 1$, see also example 1.

We consider algorithms where a sequence q and a storage capacity s is given to the input, while the number of places p and an (s, p)-processing of q is computed by the algorithm. The discussion in [RW97b] allows us to make the following assumptions about the processing of sequences q with a storage capacity of s bins. Let (q, S, j) be a configuration.

1. The bins in storage S that are destined for already open pallets will be removed automatically from S.
2. If storage S contains all bins for a pallet t, and no bin for an open pallet, then t will be opened automatically by removing the first bin for t from S.

Our stack-up algorithms will only be asked for a decision if each bin in the storage is not destined for an open pallet, and the storage does not contain all bins for some pallet. Such configurations are called *decision configurations* with respect to s. In a decision configuration (q, S, j) the stack-up algorithm decides which pallet t is opened next by removing the first bin b destined for pallet t from the storage. On-line algorithms only know the bins of q up to position $j - 1$, but additionally the number of bins for each pallet. Thus, it is possible to recognize configurations in that on-line algorithms have to open another pallet. In example 1 the first two configurations are decision configurations with respect to $s = 3$.

3 The Most-Frequently Algorithm

For a sequence of bins q and a pallet t let

$$h(q, t) := |\{\ b \in q : plt(b) = t\ \}|$$

be the number of bins for pallet t in q. The MF algorithm opens in a decision configuration (q, S, j) one of the pallets $t \in plts(S)$ for which $h(S, t)$ is maximal. If two or more pallets have the same number of bins in storage S then ties will be broken by any deterministic rule.

For a position j let $cut(q, j)$ be the set of all pallets t such that the first bin for pallet t is on a position less than j in sequence q and the last bin for pallet t is on a position greater than or equal to j in sequence q.

Consider a processing of sequence q by the MF algorithm with storage capacity s. Assume during this processing of q we get l decision configurations $(q, S_1, j_1), \ldots, (q, S_l, j_l)$ in which the MF algorithm opens the pallets $t_1, \ldots, t_l$, respectively. Then we define

$$h_{\min}(q, s) := \min\{\ h(S_1, t_1),\ \ \ldots,\ \ h(S_l, t_l),\ \ s\ \}.$$

If during the processing of q no decision configuration is reached, then $h_{\min}(q,s) = s$ by definition. Please note, that in this special case the processing takes only one stack-up place, because in each configuration the storage contains a bin for some already open pallet, or it contains all bins for some pallet.

Lemma 1. *For each* (s,p)*-sequence* q *and each integer* $c \geq 0$ *it holds*

$$p \;\geq\; \left\lceil \frac{c+1}{h_{\min}(q, s+c)} \right\rceil.$$

Proof. Consider a processing of an (s,p)-sequence q by the MF algorithm with storage capacity $s+c$. If no decision configuration is reached, $h_{\min}(q, s+c) = s+c$ by definition, and the result follows. Let (q,S,j) be a decision configuration in which the MF algorithm opens a pallet t, then the $s+c$ bins in storage S are destined for at least $\lceil \frac{s+c}{h(S,t)} \rceil$ pallets, because by the definition of the MF algorithm each pallet has at most $h(S,t)$ bins in S. Since (q,S,j) is a decision configuration, we get

$$|cut(q,j)| \;\geq\; \left\lceil \frac{s+c}{h(S,t)} \right\rceil.$$

Let A be an algorithm that yields an (s,p)-processing of sequence q. If A reaches the configuration (q,S',j) for some S' and position j as above, then

$$|open((q,S',j))| \;\geq\; \left\lceil \frac{c}{h(S,t)} \right\rceil,$$

because S' has only a capacity of s bins. If $h(S,t)$ divides c then there are at least s bins in q on a position less than j which are destined for pallets in $cut(q,j)$. In this case, at least $\frac{c}{h(S,t)} + 1$ places are necessary to continue the processing by A. This strengthen the inequality to

$$|open((q,S',j))| \;\geq\; \left\lceil \frac{c+1}{h(S,t)} \right\rceil.$$

Since $p \geq |open((q,S',j))|$ and since there is at least one decision configuration (q,S_i,j_i) in which the MF algorithm will open a pallet t_i such that $h(S_i,t_i) = h_{\min}(q,s+c)$, the result follows. □

Theorem 1. *For given integers* $s,\ p > 0$ *and* $c \geq 0$*, the MF algorithm computes an* $(s+c,p')$*-processing of any* (s,p)*-sequence* q *for some*

$$p' \;<\; p + \frac{s \cdot p}{c+1}.$$

Proof. Consider a processing of q by the MF algorithm with storage capacity $s+c$. If no decision configuration is reached, the result follows because $p' = 1$. Let (q,S,j) be a decision configuration in which a pallet t is opened and where

$$|open((q,S,j))| \;=\; p + \left\lceil \frac{s \cdot p}{c+1} \right\rceil - 2.$$

Let $T = open((q, S, j)) \cup \{t\}$. By the definition of stack-up algorithms, the pallets of T are opened in decision configurations. Therefore, for each of the $p + \lceil \frac{s \cdot p}{c+1} \rceil - 1$ pallets in T there are at least $h_{\min}(q, s+c)$ bins in q on a position less than j.

Let (q, S', j) be a corresponding configuration during an (s,p)-processing of q for some S' and position j as above. By lemma 1, we know that $h_{\min}(q, s+c) \geq \lceil \frac{c+1}{p} \rceil$. Since $\lceil \frac{s \cdot p}{c+1} \rceil \cdot \lceil \frac{c+1}{p} \rceil \geq s$, there are either $p-1$ pallets of T open in configuration (q, S', j) and another pallet of T is opened in the next transformation step, or there are p pallets of T open in (q, S', j), respectively. The storage during the next transformation steps contains always a bin for an open pallet until the last bin for some of the open pallets is removed from q. Since all these open pallets are in T, and are therefore also open during the processing by the MF algorithm, the MF algorithm removes the last bin for an open pallet during the next transformation steps, too. After that there are only $p + \lceil \frac{s \cdot p}{c+1} \rceil - 2$ pallets open and the processing continues. □

Corollary 1. *The MF algorithm takes at most* $2p$ *stack-up places and a storage capacity of* $2s$ *bins to process* (s,p)*-sequences.*

Proof. Follows immediately by theorem 1 for $c = s - 1$. □

4 Lower Bounds for On-Line Stack-Up Algorithms

For m pallets $t_1, \ldots, t_m$ let $[t_1, \ldots, t_m]$ be a sequence of m bins $(b_1, \ldots, b_m)$ such that $plt(b_i) = t_i$ for $i = 1, \ldots, m$. For two sequences of bins $q = (b_1, \ldots, b_n)$ and $q' = (b'_1, \ldots, b'_{n'})$ let $q \circ q' = (b_1, \ldots, b_n, b'_1, \ldots, b'_{n'})$. For a positive integer k and a sequence of bins q let

$$q^k = \overbrace{q \circ q \circ \cdots \circ q}^{k \text{ times}}.$$

For example, if $q \mathrel{\hat{=}} (1,2,3)$ and $q' \mathrel{\hat{=}} (4,5,6)$ then $q \circ q' \mathrel{\hat{=}} (1,2,3,4,5,6)$ and $q^3 \mathrel{\hat{=}} (1,2,3,1,2,3,1,2,3)$.

Theorem 2. *For each on-line algorithm A, each integers s, $p > 0$ and $c \geq 0$ there are (s,p)-sequences q such that A needs at least $p+m$ stack-up places to process q with a storage capacity of $s+c$ bins, where $\sum_{i=1}^{m} x_i < s$ for the following definition of x_i*

$$x_i = \begin{cases} \lceil \frac{c+1}{p} \rceil & \text{if } i = 1 \\ \lceil \frac{1}{p} \cdot (c + 1 + \sum_{j=1}^{i-1} x_j) \rceil & \text{if } i > 1. \end{cases}$$

Proof. Let $q = q_1 \circ q'_1 \circ \ldots \circ q_{m+1} \circ q'_{m+1}$ such that for $i = 1, \ldots, m$ and some $t'_i \in plts(q_i)$

$$q_i = \begin{cases} [t_{i,1}]^{x_i} \circ [t_{i,2}]^{x_i} \circ \cdots \circ [t_{i,\frac{s+c}{x_i}}]^{x_i} & \text{if } x_i \text{ divides } s+c \\ [t_{i,1}]^{x_i} \circ [t_{i,2}]^{x_i} \circ \cdots \circ [t_{i,\lfloor \frac{s+c}{x_i} \rfloor}]^{x_i} \circ [t_{i,\lceil \frac{s+c}{x_i} \rceil}]^{y_i} & \text{otherwise} \end{cases}$$

and

$$q'_i = q_i - \{t'_i\}.$$

Each q_i consists of exactly $s+c$ bins, i.e., $y_i = s+c-\lfloor \frac{s+c}{x_i} \rfloor \cdot x_i$. Let additionally

$$\begin{aligned} q_{m+1} &= [t_{m+1,1}]^{s+c} \circ \cdots \circ [t_{m+1,p}]^{s+c} \\ \text{and } q'_{m+1} &= [t_{m+1,1}, \ldots, t_{m+1,p}, t'_1, \ldots, t'_m]. \end{aligned}$$

We assume that all pallets $t_{i,j}$ are pairwise distinct. There is one such sequence q for each choice of the m pallets $t'_1, \ldots, t'_m$, and each can be processed with a storage capacity of s bins and p stack-up places by opening the pallets in the order defined by the bins in $q'_1 \circ \cdots \circ q'_{m+1}$.

Let $t_{i,l}$ be the pallet for which there is a bin in sequence q_i but no bin in sequence q'_i. Then by definition $t_{i,1}, \ldots, t_{i,l-1}, t_{i,l+1}, \ldots$ is the order of pallets defined by the bins in sequence q'_i. Consider the first decision configuration (q, S, k) after all bins for the pallets in $plts(q'_{i-1})$ have been removed from q. Then storage S contains bins for the pallets $t'_1, \ldots, t'_{i-1}$, and the first bins of sequence q_i.

Since $\sum_{j=1}^{m} x_j < s$ by assumption, and since the x_i are monotone increasing, it holds $\sum_{j=1}^{i} x_j < s$ for each $i = 1, \ldots, m$, or equivalently

$$x_i + 1 \quad \leq \quad s - \sum_{j=1}^{i-1} x_j.$$

Therefore, there are x_i bins for pallet $t_{i,1}$ in the storage of decision configuration (q, S, k), and at least one further bin for pallet $t_{i,2}$. After all bins for pallet $t_{i,1}$ have been removed from q_i, there are x_i bins for pallet $t_{i,2}$ in the storage of the next decision configuration, and at least one further bin for pallet $t_{i,3}$. Then pallet $t_{i,2}$ is opened and all bins for $t_{i,2}$ are removed from q_i. In this way the processing continues until all bins for pallet $t_{i,l-1}$ have been removed from q_i. Then at least one bin for pallet $t_{i,l+1}$ is in the storage of the next decision configuration, which can be removed. After all bins for pallet $t_{i,l+1}$ have been removed step by step from q_i, at least one bin for pallet $t_{i,l+2}$ has come into the storage. In this way the processing continues, and all pallets of $plts(q_i)$ which are also in $plts(q'_i)$ can be processed without to open any of the pallets $t'_1, \ldots, t'_i$. To see this, consider the following two cases.

If $(p+1) \cdot x_i \leq s+c$, then $p \cdot x_i$ bins have been removed from q_i by the above processing. Since

$$x_i \;\geq\; \frac{1}{p}\Big(c+1+\sum_{j=1}^{i-1} x_j\Big) \qquad \Longleftrightarrow \qquad p \cdot x_i - (c+1) \;\geq\; \sum_{j=1}^{i-1} x_j,$$

there are at least $s - \sum_{j=1}^{i-1} x_j \geq s+c+1-p \cdot x_i$ bins of sequence q_i in storage S in decision configuration (q, S, k). Thus, after $p \cdot x_i$ bins have been removed from q_i, at least the first bin b of q'_i has come into the storage.

Otherwise, if $(p+1) \cdot x_i > s+c$, then all pallets of $plts(q_i)$ except $t_{i,l}$ have been opened by the above processing, and since $\sum_{j=1}^{i} x_j \leq s-1$ the first bin b of q'_i has come into the storage. Since the pallets have been opened in the order defined by the bins in sequence q'_i, the first bins in q'_i are destined for open pallets. After their removal the next pallets can be opened.

After all bins for the pallets in $plts(q'_m)$ have been removed from q, at least one bin for pallet $t_{m+1,1}$ is in the storage of the next decision configuration, because $\sum_{j=1}^{m} x_j < s$ by assumption. Therefore, all bins for $t_{m+1,1}$ can be removed one after the other. After that all bins for $t_{m+1,2}$ can be removed one after the other. In this way the processing continues until all bins for the pallets $t_{m+1,1}, \ldots, t_{m+1,p}$ are removed from q. At the end of the processing the pallets $t'_1, \ldots, t'_m$ are opened one after the other.

Let A be an arbitrary deterministic on-line algorithm with a storage capacity of $s + c$ bins. Algorithm A opens the first pallet in some decision configuration (q, S_1, j_1), where j_1 is the position of the first bin of q'_1. Algorithm A does not know the pallet t'_1 missing in q'_1. Without loss of generality, we can assume that t'_1 is the pallet opened in configuration (q, S_1, j_1). Otherwise, we redefine q'_1 by using the first pallet t opened by algorithm A for t'_1. Since A is a deterministic on-line algorithm and $|q_1| = s + c$, pallet t opened by algorithm A is uniquely specified by sequence q_1.

The next decision configuration is (q, S_2, j_2), where j_2 is the position of the first bin of q'_2. Here again, in configuration (q, S_2, j_2) algorithm A does not know the pallet t'_2 missing in q'_2. Thus, we can assume that t'_2 is the pallet opened in configuration (q, S_2, j_2), otherwise we redefine q'_2. Generally, in decision configuration (q, S_i, j_i), where j_i is the position of the first bin in q'_i, algorithm A does not know the pallet t'_i missing in q'_i. So, we can redefine sequence q step by step such that A opens in the first m decision configurations the pallets $t'_1, \ldots, t'_m$.

By the definition of q_{m+1}, algorithm A needs p additional stack-up places and thus altogether $p + m$ stack-up places to process (s, p)-sequence q with a storage capacity of $s + c$ bins. □

5 An Optimal On-Line Stack-Up Algorithm

For some position j and a pallet $t \in cut(q, j)$ let $h(q, j, t)$ be the number of bins for pallet t which are on a position less than j in q.

Lemma 2. *Let q be an (s, p)-sequence, let (q, S_c, j) be any decision configuration with respect to $s + c$ for some given integer $c \geq 0$, and let (q, S, j) be a corresponding configuration of an (s, p)-processing of q. Let $T_c = open((q, S_c, j))$, and $T = open((q, S, j))$. Then it holds*

$$\frac{1}{p}\Big(c + \sum_{t \in T_c - T} h(q, j, t)\Big) \;\leq\; \max_{t \in T - T_c} \{\ h(q, j, t)\ \}.$$

Proof. Omitted due to space restrictions. □

Theorem 3. *For each integers s, $p > 0$, and $c \geq 0$ the MF algorithm needs at most $p + l$ stack-up places to process an (s, p)-sequence q with a storage capacity of $s + c$ bins, where $\sum_{j=1}^{l} y_j < s$ for the following definition of y_i*

$$y_i = \begin{cases} \lceil \frac{c+1}{p} \rceil & \text{if } i = 1 \\ \lceil \frac{1}{p} \cdot (c + \sum_{j=1}^{i-1} y_j) \rceil & \text{if } i > 1. \end{cases}$$

Proof. Consider some configuration (q, S_c, j) during a processing of q by the MF algorithm with a storage capacity of $s+c$ bins such that $|\mathit{open}((q, S_c, j))|$ is maximal. Let (q, S, j) be a corresponding configuration during an (s, p)-processing of q. Let $T_c = \mathit{open}((q, S_c, j))$, $T = \mathit{open}((q, S, j))$, and $T_c - T = \{t_1, \ldots, t_l\}$. Furthermore, let $(q, S_{c,1}, j_1), \ldots, (q, S_{c,l}, j_l)$ be the decision configurations in which the pallets $t_1, \ldots, t_l$, respectively, are opened by the MF algorithm, where $j_1 < \ldots < j_l$. Let $T_{c,i}$, $1 \le i \le l$, denote the set of open pallets in configuration $(q, S_{c,i}, j_i)$, and let T_i denote the set of open pallets in a corresponding configuration (q, S_i, j_i) during the (s, p)-processing of q. Then it holds

$$h(q, j_i, t_i) \ \ge \ \max_{t \in T_i - T_{c,i}} \{ \ h(q, j_i, t) \ \},$$

because the MF algorithm opens always a pallet with the most number of bins in the storage. This holds also for pallet t_i in configuration $(q, S_{c,i}, j_i)$. And by lemma 2 it holds

$$\max_{t \in T_i - T_{c,i}} \{ \ h(q, j_i, t) \ \} \ \ge \ \frac{1}{p}\Big(c + \sum_{t \in T_{c,i} - T_i} h(q, j_i, t)\Big).$$

For $l \ge i \ge k \ge 1$ it holds $h(q, j_i, t_k) \ge h(q, j_k, t_k)$ by definition. Furthermore, none of the pallets $t \in \{t_1, \ldots, t_{i-1}\}$ is open in any configuration (q, S', i'), $i' \le j_i$, during the (s, p)-processing of q, because such a pallet $t \in \mathit{open}((q, S', i'))$ would remain open up to configuration (q, S, j), and therefore pallet t has to be in T. This contradicts the assumption that the pallets $\{t_1, \ldots, t_l\}$ are not in $\mathit{open}((q, S, j))$. Thus, we get

$$h(q, j_i, t_i) \ \ge \ \frac{1}{p}\Big(c + \sum_{t \in T_{c,i} - T_i} h(q, j_i, t)\Big) \ \ge \ \frac{1}{p}\Big(c + \sum_{k=1}^{i-1} h(q, j_k, t_k)\Big).$$

Since $h(q, j_i, t_i)$ is an integer, we get $h(q, j_i, t_i) \ge \lceil \frac{1}{p}(c + \sum_{k=1}^{i-1} h(q, j_k, t_k)) \rceil$. The maximum l such that $\sum_{i=1}^{l} h(q, j_i, t_i) < s$ is an upper bound on the number of pallets which are open in configuration (q, S_c, j) and not open in configuration (q, S, j). By lemma 1 it holds $h(q, j_1, t_1) \ge \lceil \frac{c+1}{p} \rceil$. □

Corollary 2. *Most-Frequently is an optimal on-line stack-up algorithm up to at most one stack-up place.*

Proof. Let x_i and y_i be as defined in theorem 2 and 3, respectively. In theorem 2 we have seen, that for any on-line stack-up algorithm A there are (s, p)-sequences q such that A takes at least $p + m$ stack-up places to process q with a storage capacity of $s + c$ bins, where m is the largest integer that satisfies $\sum_{i=1}^{m} x_i < s$. In theorem 3 we have seen, that the MF algorithm takes at most $p + l$ stack-up places to process (s, p)-sequences with a storage capacity of $s + c$ bins, where l is the largest integer that satisfies $\sum_{i=1}^{l} y_i < s$. It can be shown that $y_i \le x_i \le y_{i+1}$ for $i \ge 1$. Therefore, m and l differ at most by one.

Theorem 4. *The number of stack-up places the MF algorithm takes to process (s,p)-sequences with storage capacity $s+c$ is less than*

$$p + p \cdot \log_2\left(\frac{s}{c+1}+1\right)+1.$$

Proof. Consider the proof of theorem 3. There we have seen that

$$h(q,j_i,t_i) \;\geq\; \left\lceil \frac{1}{p}\Big(c + \sum_{k=1}^{i-1} h(q,j_k,t_k)\Big)\right\rceil.$$

It can be shown that $(c+1)\cdot(2^{\frac{l-1}{p}}-1) \;\leq\; \sum_{i=1}^{l} h(q,j_i,t_i)$ for each $l \geq 1$. That means, if $s \;\leq\; (c+1)\cdot(2^{\frac{l-1}{p}}-1)$, then it holds $s \;\leq\; \sum_{i=1}^{l} h(q,j_i,t_i)$. Since

$$s \;\leq\; (c+1)\cdot(2^{\frac{l-1}{p}}-1) \quad\Longleftrightarrow\quad p\cdot\log_2\left(\frac{s}{c+1}+1\right)+1 \;\leq\; l,$$

the maximum number of pallets which are open in some configuration (q,S_c,j) during the processing of q by the MF algorithm and not open in configuration (q,S,j) during an (s,p)-processing is less than $p\cdot\log_2(\frac{s}{c+1}+1)+1$. In configuration (q,S_c,j) there are at most p pallets open that are open in (q,S,j), too. □

References

dK94. R. de Koster. Performance approximation of pick-to-belt orderpicking systems. *European Journal of Operational Research*, 92:558–573, 1994. 109

GJ79. M.R. Garey and D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, San Francisco, 1979. 110

LLKS93. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Handbooks in Operations Research and Management Science, vol. 4, Logistics of Production and Inventory*, pages 445–522. North-Holland, Amsterdam, 1993. 109

MMS88. M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 322–333. ACM, 1988. 110

RW97a. J. Rethmann and E. Wanke. An approximation algorithm for stacking up bins from a conveyer onto pallets. In *Proceedings of the Annual Workshop on Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 440–449. Springer-Verlag, 1997. 110

RW97b. J. Rethmann and E. Wanke. Competitive analysis of on-line stack-up algorithms. In *Proceedings of the Annual European Symposium on Algorithms*, volume 1284 of *Lecture Notes in Computer Science*, pages 402–415. Springer-Verlag, 1997. 110, 112

RW97c. J. Rethmann and E. Wanke. Storage Controlled Pile-Up Systems. *European Journal of Operational Research*, 103(3):515–530, 1997. 109, 110

By examination regulations we are obliged to notice that this work is part of an intended dissertation at the "Mathematisch-Naturwissenschaftliche Fakultät der Heinrich-Heine-Universität Düsseldorf".

On-Line Scheduling of Parallel Jobs with Runtime Restrictions

Stefan Bischof and Ernst W. Mayr

Institut für Informatik
Technische Universität München
D-80290 München, Germany
{bischof,mayr}@in.tum.de
http://wwwmayr.in.tum.de/

Abstract. Consider the execution of a parallel application that dynamically generates parallel jobs with specified resource requirements during its execution. We assume that there is not sufficient knowledge about the running times and the number of jobs generated in order to precompute a schedule for such applications. Rather, the scheduling decisions have to be made on-line during runtime based on incomplete information. We present several on-line scheduling algorithms for various interconnection topologies that use some a priori information about the job running times or guarantee a good competitive ratio that depends on the runtime ratio of all generated jobs. All algorithms presented in this paper have optimal competitive ratio up to small additive constants.

1 Introduction

The efficient operation of parallel computing systems requires the best possible use of the resources that a system provides. In order to achieve an effective utilization of a parallel machine a smart coordination of the resource demands of all currently operating applications is necessary. Consequently, the task of a scheduler is to cleverly assign the resources, most prominently the processors, to the jobs being processed. For the case of sequential jobs, i.e., jobs that require exactly one processor for execution, the involved scheduling problems have been studied intensively for decades [2]. But in many situations the problem arises to find a schedule for a set of parallel jobs [4,5,2].

The model studied in this paper assumes that each parallel job demands a fixed number of processors or a specified sub-system of a certain size and topology (depending on the underlying structure of the parallel machine considered) for its execution. It is not possible to run a parallel job on fewer processors than requested, and additional processors will not decrease the running time. This reflects the common practice that the decision on the number of processors is made before a job is passed to the scheduler based on other resource requirements like memory, disk-space, or communication intensity. The processors must be allocated exclusively to a job throughout its execution, and a job cannot be preempted or restarted later. This is a reasonable assumption because of the large overhead for these activities on parallel machines. Furthermore, there may be precedence constraints between the jobs. A job can only be executed if all of its predecessors have already completed execution. Most frequently, precedence constraints arise from data dependencies such that a job needs the complete input produced by other jobs before it can start computation.

We are concerned with on-line scheduling throughout this paper to capture the fact that complete a priori information about a job system is rarely available. However, it has been shown [6,9] that the worst-case performance of any deterministic or randomized on-line algorithm for scheduling parallel job systems with precedence constraints and arbitrary running times of the jobs is rather dismal, even if the precedence constraints between the jobs are known in advance. Therefore, we study the case that there is some a priori knowledge about the execution times of the individual jobs but the dependencies are unknown to the scheduler.

Three different gradations for this additional knowledge are studied in this paper. The first model of runtime restrictions requires that all job running times are equal and that this fact is known to the on-line scheduler. We give a level-oriented on-line algorithm for this problem that repeatedly schedules a set of available jobs using Bin Packing and collects all jobs that arrive during a phase for execution in the next phase. We show that this algorithm is 2.7-competitive if the First Fit heuristic is used. Due to a lower bound of 2.691 for every deterministic on-line scheduler (cf. [1]), our algorithm is almost optimal.

We then explore the entire bandwidth between unit and arbitrary execution times and capture the variation of the individual job running times by a characteristic parameter that we call **runtime ratio** (the quotient of the longest and shortest running time). Our second model postulates that the runtime ratio of a job system is reasonably small and that the on-line scheduler knows the shortest execution time (but not the runtime ratio itself). A family of job systems with runtime ratio $T_R \geq 2$ is given that bounds the competitive ratio of any deterministic on-line scheduler by $(T_R + 1)/2$ from below. We note that the structure of the dependency graph is an out-forest in all of our lower bound proofs. This bound remains valid even if the scheduler knows the actual runtime ratio in advance. An on-line scheduler designated RRR ($\underline{\text{R}}$estricted $\underline{\text{R}}$untime $\underline{\text{R}}$atio) for parallel systems supporting arbitrary allocations is described, and we demonstrate a competitive ratio of $T_R/2 + 4$ for this algorithm for any job system with runtime ratio $\leq T_R$. Therefore, the RRR algorithm is nearly optimal up to a small additive constant. The assumption that the shortest execution time is known to the on-line scheduler can be dropped without much loss of competitive performance. In fact, the modified algorithm for this third model is $T_R/2 + 5.5$ competitive (see [1] for details).

The remainder of this paper is organized as follows. In Section 2 we introduce our scheduling model, some notation and definitions. We then discuss previous and related work on on-line scheduling of parallel jobs in Section 3. Section 4 presents nearly optimal on-line schedulers for jobs with unit execution time, whereas in Section 5 we study job systems where the ratio of the running times of the longest and shortest job is bounded. Again, we describe and analyze on-line scheduling algorithms that are optimal up to small additive constants. Finally, we present our conclusions in Section 6.

2 Preliminaries

Let N denote the number of processors of the parallel computer-system at hand. A *(parallel) job system* is a non-empty set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_m\}$ where each job specifies the type and size of the sub-system that is necessary for its execution together with precedence-constraints among the jobs in $\mathcal{J}$ given as a partial order $\prec$ on $\mathcal{J}$. If $J_a \prec J_b$ then J_b cannot be scheduled for execution before J_a is completed. A *task* is a job that requires one processor for execution.

A *schedule* for a job system $(\mathcal{J}, \prec)$ is an assignment of the jobs to processors and start-times such that:

- each job is executed on a sub-system of appropriate type and size,
- all precedence-constraints are obeyed,
- each processor executes at most one job at any time,
- jobs are executed non-preemptively and without restarts.

The interconnection topology of the parallel computer-system may impose serious restrictions on the *job types* that can be executed efficiently on a particular machine. On a hypercube, for example, it is reasonable to execute jobs only on subcubes of a certain dimension rather than on an arbitrary subset of the processors. On the other hand, a number of interconnection networks do not restrict the allocation of processors to parallel jobs. Therefore, the various types of interconnection networks have to be treated separately. In this paper we study the *complete model* which assumes that a job J_a requests n_a processors $(1 \leq n_a \leq N)$ for execution and any subset of processors of size n_a may be allocated. Results for hypercubes can be found in [1].

It is always possible to transform a job system $(\mathcal{J}, \prec)$ into a directed acyclic graph $D = (\mathcal{J}, E)$ with $(J_a, J_b) \in E \Leftrightarrow J_a \prec J_b$. Removing all transitive edges from D we obtain the *dependency graph* induced by $(\mathcal{J}, \prec)$. We call two jobs J_a and J_b *dependent* if $J_a \prec J_b$ or $J_b \prec J_a$, and *independent* otherwise. We shall use the terms *dependency* and *precedence-constraint* interchangeably in this paper. The *length of a path* in the dependency graph induced by $(\mathcal{J}, \prec)$ is defined as the sum of the running times of the jobs along this path. A path is called *critical* if its length is maximum among all paths in the dependency graph induced by $(\mathcal{J}, \prec)$. A job is *available* if all predecessors of this job have completed execution. An on-line scheduling algorithm is only aware of available jobs and has no knowledge about their successors. We assume that the on-line scheduler receives knowledge about a job as soon as the job becomes available. This event, however, may depend on earlier scheduling decisions.

The *efficiency* of a schedule at any time t is the number of busy processors at time t divided by N. In general, the running time of a job is also unknown to the on-line scheduler and can only be determined by executing a job and measuring the time until its completion. In Section 4, though, we study the case of unit execution times and therefore restrict the on-line model there to the case of unknown precedence-constraints.

Throughout the paper we use the notations shown in Table 1 (cf. [9,6]) for a given job system $(\mathcal{J}, \prec)$:

Our goal is to generate schedules with *minimum makespan*, i.e. to minimize the completion time of the job finishing last. We evaluate the performance of our on-line scheduling algorithms by means of competitive analysis. A deterministic on-line algorithm ALG is called *c-competitive* if $T_{\text{ALG}} \leq cT_{\text{opt}}$ for **all** job systems and **arbitrary** N. The infimum of the values $c \in [1, \infty]$ for which this inequality holds is called the *competitive ratio* of ALG. The competitive ratio clearly is a worst-case measure. It is intended to compare the performance of different on-line algorithms that solve the same problem, since it is in general impossible to compute an optimal solution without complete knowledge of the problem instance. An *optimal* on-line algorithm is one with a best possible competitive ratio.

3 Previous and Related Work

Extensive work on non-preemptive on-line scheduling of parallel jobs with or without dependencies was done by FELDMANN, KAO, SGALL and TENG [6,9,7]. However, these results for general parallel job systems are bad news for users of parallel computers since they show that no deterministic on-line scheduler for N processors can have competitive ratio better than N. That is, the competitive ratio is asymptotically unbounded, and even randomization cannot improve this unsatisfactory situation substantially.

One possibility to improve the performance is to restrict the maximum job size to λN processors, $0 < \lambda < 1$. Given this restriction it has been shown that the GREEDY algorithm is optimal for the complete model with competitive ratio $1 + \frac{1}{1-\lambda}$. Setting $\lambda = 1/2$, for example, yields a 3-competitive algorithm. Another possible alternative is the use of *virtualization*. This means that a parallel job J_a which requests n_a processors is executed on a smaller number of processors n'_a by the use of simulation techniques with a predetermined increase in running time. Under the assumption of proportional slowdown (the running time of a job is enlarged by the factor n_a/n'_a) it can be shown that there is an optimal on-line scheduler for the complete model with competitive ratio $2 + \Phi$, where $\Phi = (\sqrt{5} - 1)/2$ is the golden ratio. The two approaches just described can

Table 1. Frequently used notations

T_{opt}	Length of an optimal off-line schedule for $(\mathcal{J}, \prec)$
T_{ALG}	Length of a schedule for $(\mathcal{J}, \prec)$ generated by Algorithm ALG
$T_{\max}$	Maximal length of any path in the dependency graph induced by $(\mathcal{J}, \prec)$
$t_{\min}$	Minimal running time of any job in $\mathcal{J}$
$t_{\max}$	Maximal running time of any job in $\mathcal{J}$
$T_{<\alpha}$	Total time of a schedule for $(\mathcal{J}, \prec)$ when the efficiency is less then α, $0 \leq \alpha \leq 1$

be combined to yield an optimal on-line scheduler with competitive ratio $2 + \frac{\sqrt{4\lambda^2+1}-1}{2\lambda}$ for the complete model.

Both approaches, though, have a severe drawback that arises due to the memory requirements of parallel jobs. Restricting the maximum size of a job to λN processors can thus severely restrict the problem size that can be solved on a particular machine. This is often unacceptable in practice because solving large problems is the main reason for the use of parallel computers besides solving problems fast. Virtualization may be impossible or prohibitively expensive if such memory limitations exist.

The job systems used in the lower bound proofs in [6,9] for the general case reveal an unbounded ratio of the running times of the longest and shortest job. Therefore, we think it necessary to study the influence of the individual running times on the competitive ratio of on-line schedulers for our scheduling problem. To gain insight into this relationship it is only natural to start with unit execution times as is done in Section 4. It turns out that the problem becomes manageable with small constant competitive ratio even if nothing is known about the precedence constraints.

To fill the gap between these two extremes — totally unrelated running times versus unit execution times — we identify the runtime ratio as the distinctive parameter of a job system for the achievable competitive ratio. Our results for the proposed on-line schedulers in Section 5 demonstrate a smooth, linear transition of the competitive ratio from the case of unit execution times to unrelated execution times that is governed by the runtime ratio.

4 Jobs with Unit Execution Time

In this section, we restrict our model to the case where all jobs have the same execution time. When the dependency graph is known to the scheduler this problem has been intensively studied by Garey, Graham, Johnson and Yao [8]. We show that similar results hold in an on-line environment, where a job is available only if all its predecessors have completed execution.

The Level algorithm (see Fig. 1) collects all jobs that are available from the beginning. Since available jobs are independent we can easily transform the problem of scheduling these jobs to the Bin Packing problem: the size of a job divided by N is just the size of an item to be packed, and the time-steps of the schedule correspond to the bins (see [3] for a survey on Bin Packing). Let Pack be an arbitrary Bin Packing heuristic. We parameterize the Level algorithm with Pack to express the fact that a schedule for a set of independent jobs is generated according to Pack. Thereafter, the available jobs are executed as given by this schedule. Any jobs that become available during this execution phase are collected by the algorithm. After the termination of all jobs of the first level a new schedule for all available jobs is computed and executed. This process repeats until there are no more jobs to be scheduled.

```
Algorithm LEVEL(PACK):
begin
    while not all jobs are finished do
    begin
        $\mathcal{A} := \{ J \in \mathcal{J} \mid J$ is available $\}$;    // next EPT level
        schedule all jobs in $\mathcal{A}$ according to PACK;
        wait until all scheduled jobs are finished;
    end;
end.
```

Fig. 1. The LEVEL(PACK) algorithm

We now show that the First-Fit (FF) BIN-PACKING heuristic is a good choice for PACK. FF considers all partially filled bins as possible destinations for the item to be packed. An item is placed into the first (lowest indexed) bin into which it will fit. If no such bin exists, a previously empty bin is opened and the item is placed into this bin.

Theorem 1. LEVEL(FF) *is* 2.7-*competitive.*

The proof of this theorem uses the following results from [8] involving a piecewise linear weighting function $W : [0, 1] \to [0, 8/5]$.

Lemma 1. *Let B denote a set of items with total size ≤ 1. Then*

$$\sum_{b \in B} W(\mathrm{size}(b)) \le \frac{17}{10}.$$

Theorem 2. *If L is a list of items with sizes ≤ 1, then*

$$\mathrm{FF}(L) < \sum_{x \in L} W(\mathrm{size}(x)) + 1,$$

where $\mathrm{FF}(L)$ denotes the number of bins used by First-Fit to pack L. Now we are ready to prove Theorem 1:

Proof. Let $\mathcal{J}$ be a job system with unit execution time and arbitrary dependencies. We define

$$\overline{W}(\mathcal{J}) = \sum_{j \in \mathcal{J}} W(\mathrm{size}(j)).$$

Thus $\overline{W}(\mathcal{J})$ is the total weight of all job sizes. Let l be the number of levels of the job system. For $1 \le i \le l$ let U_i be the set of jobs of each level. By Theorem 2 we can upper bound the length of the partial schedule for each level i, $1 \le i \le l$, generated by LEVEL(FF):

$$T_{\text{LEVEL(FF)}}(U_i) < \overline{W}(U_i) + 1.$$

We can think of an optimal packing of $\mathcal{J}$ with the dependencies removed as a partition of $\mathcal{J}$ into $\mathcal{J}^*$ sets each of which has total size ≤ 1. Applying Lemma 1 yields $\overline{W}(\mathcal{J}) \le \frac{17}{10}\mathcal{J}^*$. Together with the fact that the length of the optimal schedule for $\mathcal{J}$ without dependencies cannot be longer than the length of the optimal schedule for $\mathcal{J}$ we conclude:

$$T_{\textsc{Level(FF)}} = \sum_{i=1}^{l} T_{\textsc{Level(FF)}}(U_i) < \sum_{i=1}^{l} (\overline{W}(U_i) + 1) = \overline{W}(\mathcal{J}) + l \leq 1.7\, T_{\mathrm{opt}} + l.$$

Since $l = T_{\max} \leq T_{\mathrm{opt}}$, the result follows. □

Note that the results of this section remain valid if we assume a 1-dimensional array of length N as interconnection topology instead of using the complete model, since the Bin Packing algorithms assign consecutive processors to the jobs and the assignments in different time-steps are independent from each other.

5 Parallel Job Systems with Restricted Runtime Ratio

We have shown in the preceding section that on-line scheduling of parallel jobs with unit execution time and precedence-constraints is possible with small constant competitive ratio. On the other hand, if execution times are arbitrary, there exists no on-line scheduler with acceptable worst-case performance. It is only natural to explore the case that job runtimes are restricted by some criterion other than unit execution time in order to achieve a respectable competitive ratio. For a set of jobs $\mathcal{J}$ we therefore define the *runtime ratio* $\mathrm{RR} = \mathrm{RR}(\mathcal{J}) := t_{\max}/t_{\min}$.

In this section we study the problem of on-line scheduling parallel job systems with dependencies where the runtime ratio is bounded from above by a parameter $T_R \geq 1$ which is not known to the on-line scheduler. This problem often arises in practice when upper and lower bounds for the running time of a job are known in advance but the actual running time is unknown. This situation also makes clear that the parameter T_R cannot be used as additional information for scheduling decisions by the on-line scheduler and is therefore not part of the problem instance. Indeed, our results show that this knowledge is not necessary for the on-line scheduler to achieve a near optimal competitive ratio that depends only on T_R.

First, we give a lower bound of $(T_R + 1)/2$ for the asymptotic competitive ratio of any deterministic on-line scheduler for this problem. For simplicity we normalize the running time of the shortest job to 1. The job system used in this lower bound argument is very simple and consists of N layers with two tasks and one parallel job of size N on each layer. The parallel job depends on one of the tasks on the same layer and is predecessor of both tasks of the following layer. The task scheduled first by the on-line scheduler is assigned running time T_R and the remaining task runs for 1 unit of time and is predecessor of the parallel job. Clearly, the makespan of any schedule generated by an on-line scheduler is at least $N(T_R + 1)$. If T_R is sufficiently large (e.g., $T_R \geq 2$), the optimal solution first schedules the critical path which has length $2N$ followed by the tasks of length T_R in parallel.

The competitive ratio of any deterministic on-line scheduler is thus lower bounded by

$$\frac{N(T_R+1)}{2N+T_R} \quad \xrightarrow[N\to\infty]{} \quad \frac{T_R+1}{2}.$$

```
Algorithm RRR
begin
    while $L_1 \neq \emptyset$ do schedule a big job exclusively;
    while not all jobs are finished do
    begin
        while $L_2 \neq \emptyset$ do schedule small jobs greedily;
        if $L_1 \neq \emptyset$ then
            if a big job can be scheduled then do it;
            elsif $\alpha \geq 1/2$ then
                wait for a scheduled job to finish;
            else   // start of a delay phase
            begin
                collect small jobs that become available during the next 2 units of time;
                schedule those jobs greedily and then wait for all scheduled jobs to finish;
                while $L_1 \neq \emptyset$ do schedule a big job exclusively;
            end;
        else wait for next available job;
    end;
end.
```

Fig. 2. The RRR algorithm

We now describe an algorithm designated RRR (see Fig. 2) that achieves competitive ratio $T_R/2+4$. A key feature of this algorithm is the distinction between *big* jobs that request more than half of the total number of processors and *small* jobs with size $\leq \lfloor N/2 \rfloor$. Let $\alpha := \alpha(t)$ denote the efficiency at time t. The RRR algorithm tries to keep the efficiency at least 1/2 whenever possible. There are two reasons that hinder the RRR algorithm from achieving this goal. First, there might be no job available and second, there might be not enough processors available to schedule a big job. Therefore, the RRR algorithm must prevent big jobs from being delayed too long in order to bound the fraction of the total schedule length with low efficiency. This is done by occasionally stopping to schedule small jobs, if all big jobs request more processors than currently available and the efficiency is below 1/2.

We present two versions of the RRR algorithm. The first one assumes that $t_{\min}$ is a known quantity. Again, we normalize the running time of the shortest job to 1 and a *unit of time* refers to this normalized time quantum. In the second version we remove this assumption and employ an adaptive waiting-strategy to maintain a comparable competitive ratio. The RRR algorithm maintains two sets, L_1 and L_2, containing the available big respectively small jobs. We assume that any job that becomes available is immediately inserted into the appropriate set, and we will not state this activity explicitly in the pseudo-code description of the RRR algorithm.

Theorem 3. *The* RRR *algorithm is* $(T_R/2+4)$*-competitive for any job system* $(\mathcal{J}, \prec)$ *and* $\mathrm{RR}(\mathcal{J}) \leq T_R$.

Proof. We partition the schedule generated by the RRR algorithm into 3 different kinds of phases:

1. Efficiency is at least 1/2.
2. Efficiency is below 1/2 and there is no job available.
3. Efficiency is below 1/2 and the algorithm waits for the termination of all jobs.

We refer to the third type as a *delay phase* and denote the total time of each kind by $T_{\geq 1/2}$, T_{nojob} , and T_{delay} respectively. The total time of the RRR schedule that is spent in phases of type 1 and 2 can easily be bounded by $3\,T_{\text{opt}}$, because we have $T_{\geq 1/2} \leq 2\,T_{\text{opt}}$ by a straightforward area-argument and $T_{\text{nojob}} \leq T_{\max} \leq T_{\text{opt}}$ by GRAHAM's critical path argument.

It remains to show that $T_{\text{delay}} \leq (T_R/2 + 1)T_{\text{opt}}$. We define a *delayed job* as a big job that was available at the beginning of a delay phase. Let t_i denote the start time of delay phase i. First, we bound the length of a delay phase by $T_R + 2$. If no small jobs become available during the first two units of time after the beginning of a delay phase, no more jobs are scheduled until all currently running jobs terminate. Since the running time of any job is no more than T_R , such a delay phase lasts at most time T_R. On the other hand, if small jobs become available during the first two units of time, then these are collected and scheduled greedily at time $t_i^s = t_i + 2$ (resp. $t_i^s < t_i + 2$ if all jobs running at time t_i terminate before two units of time have elapsed) in addition to those jobs still running at time t_i^s. If the total size of these small jobs is no more than the number of idle processors at time t_i^s, they can be scheduled immediately. Clearly, the length of a delay phase is bounded by $T_R + 2$ in this case. Should the total size of the small jobs exceed the number of idle processors at time t_i^s we can schedule enough small jobs to raise the efficiency above 1/2 as long as small jobs that were collected during the interval $[t_i, t_i^s]$ are available. The time-span while the efficiency is at least 1/2 is, of course, a phase of type 1 and not part of the delay phase. Clearly, the length of the second part of a delay phase is bounded by T_R and therefore the length of a delay phase is always bounded by $T_R + 2$.

Let d denote the number of delay phases in a schedule generated by the RRR algorithm. We distinguish two cases:

1. $d = 1$: We have to show that the optimal solution needs at least time 2. This follows immediately from the fact that each delayed job must have a predecessor in the job system because otherwise it would have been scheduled earlier.
2. $d > 1$: This case will be proven by constructing a chain of jobs in the dependency graph with total execution time at least $2d$. From that we have $T_{\text{opt}} \geq 2d$ and together with $T_{\text{delay}} \leq d(T_R + 2)$ the claim follows.

The construction of this chain proceeds as follows: Starting with an arbitrary delayed job that is scheduled after delay phase d we observe that there must be a small job that is ancestor of this delayed job and is scheduled immediately after the delayed jobs of delay phase $d - 1$ (i.e. without having a small job as direct predecessor that is itself scheduled after the delayed jobs of delay phase $d - 1$)

because otherwise this delayed job would have been scheduled earlier. We add such a small job at the front of the chain. To augment the chain, we state the possibilities for the direct predecessor of a small job that is scheduled by the RRR algorithm immediately after the delayed jobs of delay phase i:

Type 1: Delayed job of delay phase i or big job that is successor of a delayed job of delay phase i,
Type 2: Small job collected during delay phase i,
Type 3: Small job running from the beginning of delay phase i.

This is due to the fact that the RRR algorithm schedules all small jobs that are available by time t_i^s before the delayed jobs of delay phase i.

We continue the construction inductively according to these three possibilities. If there is a direct predecessor of Type 1 of the small job that is currently head of the list, we can repeat the initial construction step of the chain and add a delayed job and its small ancestor at the front of the chain. When there is no direct predecessor of Type 1 but a direct predecessor of Type 2, we add 2 more jobs at the front of the chain: the Type 2 job and a direct predecessor of this job that was running at the beginning of the delay phase during which this Type 2 job was collected. Finally, if there is only a direct predecessor of Type 3, we add this job at the front of the chain. The inductive construction stops as soon as the head of the chain is a small job that is scheduled before the delayed jobs of the first delay phase.

To complete the proof, we show that the total execution time of the jobs along this chain is at least $2d$. The construction of the chain starts with 2 jobs, a delayed job and its small ancestor. Since the minimum running time of any job is 1, these 2 jobs need at least 2 units of time for execution in any schedule. If the construction proceeds by adding a Type 1 job, the same argument applies. Continuing with a Type 2 job means that again 2 more jobs with were added to the chain. If a Type 3 job is encountered, we know that this job must have execution time at least 2 because it is direct predecessor of a small job that is scheduled immediately after the delayed jobs of the delay phase the Type 3 job belongs to. Thus, for each delay phase in the schedule generated by the RRR algorithm, the above construction adds jobs with total execution time at least 2 to the chain. □

The assumption that $t_{\min}$ is known to the RRR algorithm can be dropped by employing an adaptive waiting strategy without much loss in competitive performance:

Theorem 4. *The* RRR_ADAPTIVE *algorithm is* $(T_R/2 + 5.5)$*-competitive for any job system* $(\mathcal{J}, \prec)$ *and* $\mathrm{RR}(\mathcal{J}) \leq T_R$.

Proof. Omitted due to space restrictions (see [1]). □

6 Conclusion

We have presented and analyzed several on-line scheduling algorithms for parallel job systems. It has become evident that *runtime restrictions* improve the competitive performance achievable by on-line schedulers. Therefore, if enough a priori knowledge on job running times is available to bound the runtime ratio of a job system, our schedulers can guarantee reasonable utilization of the parallel system. But even without any such knowledge the RRR_Adaptive algorithm produces schedules that are almost best possible from a worst-case point of view. All on-line algorithms considered in this paper are computationally simple, and thus the scheduling overhead involved can safely be neglected, provided that the system has suitable means to deliver the necessary load information.

References

1. Stefan Bischof and Ernst W. Mayr. On-Line Scheduling of Parallel Jobs with Runtime Restrictions. SFB-Bericht 342/04/98 A, SFB 342, Institut für Informatik, Technische Universität München, April 1998. http://wwwmayr.in.tum.de/berichte/1998/TUM-I9810.ps.gz. 120, 121, 128
2. J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes.* Springer-Verlag, Berlin, 1996. 119
3. E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation Algorithms for Bin Packing: A Survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 2, pages 46–93. PWS Publishing Company, Boston, 1996. 123
4. Dror G. Feitelson and Larry Rudolph. Parallel Job Scheduling: Issues and Approaches. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (IPPS' 95 Workshop)*, LNCS 949, pages 1–18, Berlin, 1995. Springer-Verlag. 119
5. Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and Practice in Parallel Job Scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (IPPS' 97 Workshop)*, LNCS 1291, pages 1–34, Berlin, 1997. Springer-Verlag. 119
6. Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. Optimal Online Scheduling of Parallel Jobs with Dependencies. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 642–651, New York, 1993. 120, 122, 123
7. Anja Feldmann, Jiří Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science, Special Issue on Dynamic and On-line Algorithms*, 130(1):49–72, 1994. 122
8. M.R. Garey, R.L. Graham, D.S. Johnson, and A.C.-C. Yao. Resource Constrained Scheduling as Generalized Bin Packing. *J. Comb. Theory Series A*, 21:257–298, 1976. 123, 124
9. Jiří Sgall. *On-Line Scheduling on Parallel Machines.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1994. 120, 122, 123

Testing the Quality of Manufactured Disks and Cylinders*

Prosenjit Bose and Pat Morin

School of Computer Science, Carleton University
{jit,morin}@scs.carleton.ca

Abstract. We consider the problem of testing the roundness of a manufactured object using the finger probing model of Cole and Yap [1]. When the object being tested is a disk and it's center is known, we describe a procedure which uses $O(n)$ probes and $O(n)$ computation time. (Here $n = |1/q|$, where q is the quality of the object.) When the center of the object is not known, a procedure using $O(n)$ probes and $O(n \log n)$ computation time is described. When the object being tested is a cylinder of length l, a procedure is described which uses $O(ln^2)$ probes and $O(ln^2 \log ln)$ computation time. Lower bounds are also given which show that these procedures are optimal in terms of the number of probes used.

1 Introduction

The field of metrology is concerned with measuring the quality of manufactured objects. A basic task in metrology is that of determining whether a given manufactured object is of acceptable quality. Usually this involves probing the surface of the object using a measuring device such as a coordinate measuring machine to get a set S of sample points, and then verifying, algorithmically, how well S approximates an ideal object.

A special case of this problem is determining whether an object is *round*, or *circle like*. For our purposes, an object I is *good* if the boundary of I can be contained in an annulus of inner radius $1 - \epsilon$ and outer radius $1 + \epsilon$, for some quality parameter $\epsilon > 0$, and is *bad* otherwise. See Fig. 1 for examples of good and bad objects. We call this problem the *roundness classification problem.*

Little research has been done on probing strategies for the roundness classification problem. A notable exception is the work by Mehlhorn, Shermer, and Yap [4], in which a probing strategy for manufactured disks is coupled with a roundness testing algorithm. Unfortunately, the procedure described in [4] relies on the assumption that the object I is convex. It is usually not the case that the manufacturing process can guarantee this.

In this paper we describe strategies for testing the roundness of manufactured disks and cylinders. We use the *finger probing* model of Cole and Yap [1]. In this model, the measurement device can identify a point in the interior of I and can

* This work was funded in part by the Natural Sciences and Engineering Research Council of Canada and by ALMERCO Inc.

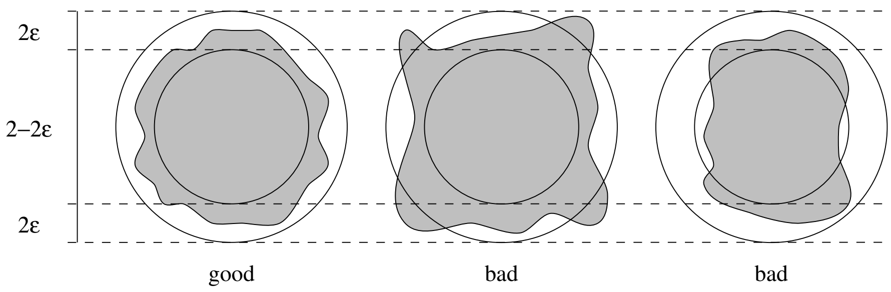


Fig. 1. Examples of good and bad objects.

probe along any ray originating outside of I, i.e., determine the first point on the ray which intersects the boundary of I. The finger probing model is a reasonable abstract model of a coordinate measuring machine [5].

This work extends the results of [4] in several ways. The assumption that the object I is convex is replaced by a much weaker assumption related to visibility. Using this assumption, we give a procedure for testing the roundness of a manufactured disk I using $O(|1/\mathrm{qual}(I)|)$ probes and $O(|1/\mathrm{qual}(I)| \log |1/\mathrm{qual}(I)|)$ computation time. Here $|\mathrm{qual}(I)|$ measures how far the object I is from the boundary between good and bad. For testing the roundness of a manufactured cylinder J we describe a procedure that uses $O(l/\mathrm{qual}(J)^2)$ probes and $O(l/\mathrm{qual}(J)^2 \log(l/\mathrm{qual}(J)^2))$ computation time, where l is the length of J. We also give lower bounds which show that our procedures are optimal, up to constant factors, in terms of the number of probes used.

The remainder of the paper is organized as follows: Section 2 introduces definitions and notation used throughout the remainder of the paper. Section 3 describes procedures for testing the quality of manufactured disks. Section 4 presents a procedure for testing the quality of manufactured cylinders. Section 5 gives lower bounds on the number of probes needed to solve these problems.

2 Definitions, Notation, and Assumptions

In this section, we introduce definitions and notation used throughout the remainder of this paper, and state the assumptions we make on the object being tested. For the most part, notation and definitions are consistent with [4].

For a point p, we use the notation $\mathrm{x}(p)$, $\mathrm{y}(p)$, and $\mathrm{z}(p)$ to denote the x, y, and z coordinates of p, respectively. The letter O is used to denote the origin of the coordinate system. We use the notation $\mathrm{dist}(a, b)$ to denote Euclidean distance between two objects. When a and b are sets of points, $\mathrm{dist}(a, b)$ is the minimum distance between all pairs of points in a and b. The angle formed by three points a, b, and c, is denoted by $\angle abc$, and we always mean the smaller angle unless stated otherwise.

A *planar object* I is defined to be any compact simply connected subset of the plane, with boundary denoted by $\mathrm{bd}(I)$. For a point p, we use $R(p, I)$ and $r(p, I)$ to denote the maximal and minimal distance, respectively, from p to a point in $\mathrm{bd}(I)$. I.e.,

$$R(p, I) = \max\{\mathrm{dist}(p, p') : p' \in \mathrm{bd}(I)\} \tag{1}$$

$$r(p, I) = \min\{\mathrm{dist}(p, p') : p' \in \mathrm{bd}(I)\} \ . \tag{2}$$

For a point p, let

$$\mathrm{qual}(p, I) = \min\{r(p, I) - (1 - \epsilon), (1 + \epsilon) - R(p, I)\} \tag{3}$$

$$\mathrm{qual}(I) = \max\{\mathrm{qual}(p, I) : p \in \mathbb{R}^2\} \ . \tag{4}$$

Any point c_I with $\mathrm{qual}(c_I, I) = \mathrm{qual}(I)$ is called a *center* of I. The value $\mathrm{qual}(I)$ is called the *quality* of the object I. An object I with $\mathrm{qual}(I) > 0$ is *good* while an object I with $\mathrm{qual}(I) < 0$ is *bad*. A procedure which determines whether a planar object is good or bad is called a *roundness classification procedure*.

In order to have a testing procedure which is always correct and which terminates, it is necessary to make some assumptions about the object I being tested. The following assumption made in [4] is referred to as the *minimum quality assumption*, and refers to the fact that the manufacturing process can guarantee that manufactured objects have a minimum quality (although perhaps not enough to satisfy our roundness criterion).

Assumption 1. $R(c_I, I) \leq 1 + \delta$ and $r(c_I, I) \geq 1 - \delta$, for some constant $0 < \delta < 1/21$, i.e., the boundary of I is contained in an annulus of inner radius $1 - \delta$ and outer radius $1 + \delta$.

The minimum quality assumption alone is not sufficient. If the object under consideration contains oddly shaped recesses, then it may be the case that these recesses cannot be found using finger probes. We say that an object I is *star-shaped* if there exists a point $k \in I$ such that for any point $p \in I$, the line segment joining k and p is a subset of I. We call the set of all points with this property the *kernel* of I. The following assumption ensures that all points in $\mathrm{bd}(I)$ can be probed by directing probes close to the center of I.

Assumption 2. I is a star-shaped object, and its kernel contains all points p such that $\mathrm{dist}(c_I, p) \leq \alpha$, for some constant $1 - \delta > \alpha > 2\delta$.

We observe that our assumptions are weaker than those in [4].

Observation 1. *The set of convex objects satisfying Assumption 1 is strictly contained in the set of objects satisfying Assumptions 1 and 2.*

3 Testing Disks

3.1 The Simplified Procedure

In this section we describe a simplified testing procedure which assumes that we know the object being tested is centered at the origin, O. The motivation for describing this simplified procedure is pedagogical; it is a simple example which helps in understanding the full procedure.

Our testing procedure tests the roundness of an object I by taking a set S of probes at uniform intervals directed at the origin. We use the notation probe(n, p) to denote the set of points obtained by taking n probes directed at the point p in directions $2\pi/n, 4\pi/n, \ldots, 2(n-1)\pi/n$. The procedure repeatedly doubles the size of the sample until either (1) a set of sample points is found which cannot be covered by an annulus of inner radius $1-\epsilon$ and outer radius $1+\epsilon$, in which case I is rejected, or (2) the set of sample points can be covered by an annulus with inner radius sufficiently larger than $1-\epsilon$ and outer radius sufficiently smaller than $1+\epsilon$, in which case we can be sure that I is a good object.

Procedure 1 Tests the roundness of the object I centered at the origin.

```
1: $r \leftarrow 1$
2: $R \leftarrow 1$
3: $n \leftarrow n_0$
4: $\Delta \leftarrow f(n)$
5: **repeat**
6:    $S \leftarrow \mathrm{probe}(n, O)$
7:    **if** $\exists p \in S : \mathrm{dist}(p, O) > 1 + \epsilon$ or $\mathrm{dist}(p, O) < 1 - \epsilon$ **then**
8:       **return** REJECT
9:    **end if**
10:   $r \leftarrow 1 - \epsilon + \Delta$
11:   $R \leftarrow 1 + \epsilon - \Delta$
12:   $n \leftarrow 2n$
13:   $\Delta \leftarrow f(n)$
14: **until** $\forall p \in S : \mathrm{dist}(p, O) < R$ and $\mathrm{dist}(p, O) > r$
15: **return** ACCEPT
```

The function $f(n)$ which appears in the procedure is defined as

$$f(n) = \frac{1}{n}\left(\frac{(1+\delta)^2(\alpha^2+1+2\delta+\delta^2)}{\alpha^2}\right)^{\frac{1}{2}} \le \frac{12}{n} \ . \tag{5}$$

and the constant n_0 is defined as

$$n_0 = \lceil \pi / \arctan(\alpha/(1+\delta)) \rceil \le 70 \ . \tag{6}$$

With these definitions, we obtain the following crucial lemma.

Lemma 1. *Let I be a planar object with center c_I. Let S be the set of results of $n \ge n_0$ probes directed at c_I in directions $0, 2\pi/n, 4\pi/n, \ldots, 2\pi(n-1)/n$. Then for any point $p \in \mathrm{bd}(I)$, there exists a point $p' \in S$ such that $\mathrm{dist}(p, p') \le f(n)$.*

Proof. Assume wlog that $c_I = O$, $\mathrm{x}(p) = 0$, and $1 - \delta \le \mathrm{y}(p) \le 1 + \delta$. We will upper-bound $|\mathrm{x}(p) - \mathrm{x}(p')|$ and $|\mathrm{y}(p) - \mathrm{y}(p')|$. Refer to Fig. 2 for an illustration.

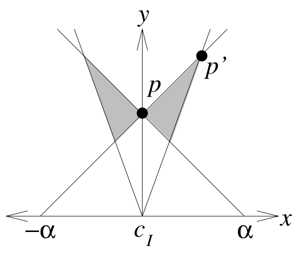


Fig. 2. Constraints on the position of p'. The point p' must be in the shaded region, and $\mathrm{dist}(p, p')$ is maximized when p' is placed as shown.

First note that there exists a sample point $p' \in S$ such that $0 \leq \angle p c_I p' \leq \pi/n$. By Assumption 1, $\mathrm{dist}(O, p') \leq 1 + \delta$, so an upper bound on $|\mathrm{x}(p) - \mathrm{x}(p')|$ is

$$|\mathrm{x}(p) - \mathrm{x}(p')| = |\mathrm{x}(p')| \leq (1+\delta)\sin(\pi/n) \tag{7}$$

$$\leq (1+\delta)\pi/n \tag{8}$$

Since $\angle p c_I p' \leq \pi/n$, p' must lie in the cone defined by the inequality

$$\mathrm{y}(p') \geq |\mathrm{x}(p')| \left(\frac{\cos(\pi/n)}{\sin(\pi/n)}\right) \quad . \tag{9}$$

Next we note that the slope of the line through p' and p must be in the range $[-\mathrm{y}(p)/\alpha, \mathrm{y}(p)/\alpha]$, otherwise Assumption 2 is violated. If $n \geq n_0$, then the region in which p' can be placed is bounded, and $|\mathrm{y}(p) - \mathrm{y}(p')|$ is maximized when p' lies on one of the bounding lines

$$f_l(x) = x\mathrm{y}(p)/\alpha + \mathrm{y}(p) \tag{10}$$

$$f_r(x) = -x\mathrm{y}(p)/\alpha + \mathrm{y}(p) \tag{11}$$

Since both lines are symmetric about $x = 0$, assume that $\mathrm{x}(p')$ lies on f_l, giving

$$|\mathrm{y}(p) - \mathrm{y}(p')| \leq |\mathrm{y}(p) - f_l(\mathrm{x}(p'))| \tag{12}$$

$$= |\mathrm{x}(p')\mathrm{y}(p)/\alpha| \tag{13}$$

$$\leq |\mathrm{x}(p')(1+\delta)/\alpha| \tag{14}$$

$$\leq (1+\delta)^2\pi/\alpha n \tag{15}$$

Substituting (8) and (15) into the Euclidean distance formula and simplifying yields the stated inequality. □

Theorem 1. *There exists a roundness classification procedure that can correctly classify any planar object I with center $c_I = O$ and satisfying Assumptions 1 and 2 using $O(|1/\mathrm{qual}(I)|)$ probes and $O(|1/\mathrm{qual}(I)|)$ computation time.*

Proof. We begin by showing that the procedure is correct. We need to show that the procedure never rejects a good object and never accepts a bad object. The former follows from the fact that the procedure only ever rejects an object when it finds a point on the object's boundary which is not contained in the annulus of inner radius $1-\epsilon$ and outer radius $1+\epsilon$ centered at c_I.

Next we prove that the procedure never accepts an bad object. Lemma 1 shows that there is no point in $\mathrm{bd}(I)$ which is of distance greater than $f(n)$ from all points in S. The procedure only accepts I when all points in S are of distance at least $\Delta = f(n)$ from the boundary of the annulus of inner radius $1-\epsilon$ and outer radius $1+\epsilon$ centered at O. Therefore, if the procedure accepts I, all points in $\mathrm{bd}(I)$ are contained in an annulus of inner radius $1-\epsilon$ and outer radius $1+\epsilon$, i.e., the object is good.

Next we prove that the running time is $O(|1/\mathrm{qual}(I)|)$. First we observe that $f(n) \in O(1/n)$. Next, note that the computation time and number of probes used during each iteration is linear with respect to the value of n, and the value of n doubles after each iteration. Thus, asymptotically, the computation time and number of probes used are dominated by the value of n during the last iteration. There are two cases to consider.

Case 1: Procedure 1 accepts I. In this case, the procedure will certainly terminate once $\Delta \leq \mathrm{qual}(I)$. This takes $O(\log(1/\mathrm{qual}(I)))$ iterations. During the final iteration, $n \in O(1/\mathrm{qual}(I))$.

Case 2: Procedure 1 rejects I. In this case, there is a point on $\mathrm{bd}(I)$ at distance $\mathrm{qual}(I)$ outside the circle with radius $1+\epsilon$ centered at O, or there is a point in $\mathrm{bd}(I)$ at distance $\mathrm{qual}(I)$ inside of the circle with radius $1-\epsilon$ centered at O. In either case, Lemma 1 ensures that the procedure will find a bad point within $O(\log|1/\mathrm{qual}(I)|)$ iterations. During the final iteration, $n \in O(|1/\mathrm{qual}(I)|)$. □

3.2 The Full Procedure

The difficulty in implementing Procedure 1 is that we may not know the position of the exact center, c_I, of I. However, the following result from [4] allows us to use this procedure anyhow.

Theorem 2 (Near-Center). *Let I be a planar object with center c_I and which satisfies Assumption 1. Then 6 probes and constant computation time suffice to determine a point c_0 such that* $\mathrm{dist}(c_I, c_0) \leq 2\delta$.

We call any such point c_0 a *near-center* of I. As the following lemmata show, knowing a near center is almost as useful as knowing the true center. Before we state the lemma, we need the following definitions.

$$f'(n) = \frac{1}{n}\left((1+3\delta)^2\pi^2 + \frac{(1+3\delta)^4\pi^2}{(\alpha-2\delta)^2}\right)^{\frac{1}{2}} \tag{16}$$

$$n'_0 = \lceil \pi/\arctan(\alpha/(1+3\delta)) \rceil \tag{17}$$

Lemma 2. *Let I be a planar object with center c_I and near-center c_0. Let S be the set of results of $n \geq n_0'$ probes directed at c_0 in directions $0, 2\pi/n, 4\pi/n, \ldots,$ $2\pi(n-1)/n$. Then for any point $p \in \mathrm{bd}(I)$, there exists a point $p' \in S$ such that* $\mathrm{dist}(p, p') \leq f'(n)$.

Proof. The proof is almost a verbatim translation of the proof of Lemma 1, except that we assume that $c_0 = O$. With this assumption we derive the bounds

$$|\mathrm{x}(p) - \mathrm{x}(p')| \leq (1 + 3\delta)(\pi/n) \tag{18}$$

$$|\mathrm{y}(p) - \mathrm{y}(p')| \leq (1 + 3\delta)^2 \pi/n(\alpha - 2\delta)| \tag{19}$$

Substituting these values into the formula for the Euclidean distance and simplifying yields the desired result. □

Lemma 3. *Let I be a planar object with center c_I and near-center c_0. Let S be the set of results of n probes directed at c_0 in directions $0, 2\pi/n, 4\pi/n, \ldots,$ $2\pi(n-1)/n$, and let c_S be the center of S. Then*

$$R(c_S, S) \leq R(c_S, I) \leq R(c_S, S) + f'(n) \tag{20}$$

$$r(c_S, S) - f'(n) \leq r(c_S, I) \leq r(c_S, S) \ . \tag{21}$$

Proof. We prove only the bounds on the $R(c_S, I)$ as the proof of the bounds on $r(c_S, I)$ are symmetric. The lower bound on $R(c_S, I)$ is immediate, since $S \subset \mathrm{bd}(I)$. To see the upper bound, choose any point $p \in \mathrm{bd}(I)$ such that $\mathrm{dist}(c_S, p) = R(c_S, I)$. By Lemma 1 there exists $p' \in S$ such that $\mathrm{dist}(p', p) \leq f'(n)$. Therefore $\mathrm{dist}(c_S, p) \leq \mathrm{dist}(c_S, p') + f'(n)$, which implies that $R(c_S, I) \leq R(c_S, S) + f'(n)$. □

Lemma 4. *Let I be a planar object with center c_I and near-center c_0. Let S be the set of results of n probes directed at c_0 in directions $0, 2\pi/n, 4\pi/n, \ldots,$ $2\pi(n-1)/n$. Then* $\mathrm{qual}(S) - f'(n) \leq \mathrm{qual}(I) \leq \mathrm{qual}(S)$

Proof. The proof can be found in [4].

Theorem 3. *There exists a roundness classification procedure that can correctly classify any planar object I satisfying Assumptions 1 and 2 using $O(1/|\mathrm{qual}(I)|)$ probes and $O(|1/\mathrm{qual}(I)| \log |1/\mathrm{qual}(I)|)$ computation time.*

Proof. We make the following modifications to Procedure 1. In Line 3, we set the value of n to n_0'. In Lines 4 and 13, we replace $f(n)$ with $f'(n)$. In Line 6 we directed our probes at c_0 rather than O. In Lines 7 and 14, we replace the simple test with a call to one of the $O(n \log n)$ time referenced roundness algorithms in [2] or [3], to test whether the sample set S can be covered by an annulus with the specified inner and outer radius.

Lemma 4 ensures that the procedure never accepts a bad object and never rejects a good object. i.e., the procedure is correct. The procedure terminates once $f'(n) < |\mathrm{qual}(I)|$. This happens after $O(\log |1/qual(I)|)$ iterations, at which point $n \in O(|1/\mathrm{qual}(I)|$. □

4 Testing Cylinders

Before we can describe a quality testing procedure for cylinders, we must generalize the notion of quality to cylinders. In this work, we are concerned with the roundess of cylinders, and not their height, or the flatness of their ends. For these reasons, we will assume that our manufactured cylinders have length l, and that their ends are perfectly flat. The object, J, that we are interested in testing is a compact simply connected subset of the space $(x, y, [0, l])$.

We assume that J is resting on the (x, y) plane and that we know the orientation of the cylinder. Define J_h to be the set of all points (x, y) such that $(x, y, h) \in J$. Note that J_h is a planar object. We define the *outer boundary* of J as $J_{\text{out}} = \bigcup_{0 \le h \le_l} J_h$, and we define the *inner boundary* of J as $J_{\text{in}} = \bigcap_{0 \le h \le_l} J_h$. For a point p on the plane, we use $R(p, J)$ and $r(p, J)$ to denote the maximal and minimal distance, respectively, from p to a point in $\text{bd}(J_{\text{out}})$ and $\text{bd}(J_{\text{in}})$, respectively. For a point p, let

$$\text{qual}(p, J) = \min\{r(p, J) - (1 - \epsilon), (1 + \epsilon) - R(p, J)\} \tag{22}$$

$$\text{qual}(J) = \max\{\text{qual}(p, J) : p \in \mathbb{R}^3\} \ . \tag{23}$$

We call any point c_J with $\text{qual}(c_J, J) = \text{qual}(J)$ a *center* of J. Note that according to these definitions J is an good object if there exists an annulus of inner radius $1 - \epsilon$ and outer radius $1 + \epsilon$ which covers both $\text{bd}(J_{\text{in}})$ and $\text{bd}(J_{\text{out}})$.

We require the following minimum quality assumptions.

Assumption 3. $R(c_J, J) \le 1 + \delta$ and $r(c_J, J) \ge 1 - \delta$, for some constant $0 < \delta < 1/21$.

Assumption 4. For all $0 \le h \le l$, J_h is a star-shaped object, and its kernel contains all points p such that $\text{dist}(c_J, p) \le \alpha$, for some constant $1 - \delta > \alpha > 2\delta$.

Assumption 5. Let J be an object with center c_J, and let $J[h-\alpha, h+\alpha]$ be the subset of points in J with z coordinate in $[h-\alpha, h+\alpha]$. Let K be the intersection of the infinite length cylinder of radius α, which is perpendicular to the (x, y) plane and is centered at O with $J[h - \alpha, h + \alpha]$. Then $J[h - \alpha, h + \alpha]$ is a star shaped object with kernel K.

Note that Assumptions 3 and 4 allow us to find a near-center c_0 of J in constant time. Our procedure for testing cylinders is exactly the same as the procedure for testing disks described in Sect. 3.2, except that during each iteration, we perform $ln/2\pi$ sets of probes along the planes $z = 0, z = 2\pi/n, z = 4\pi/n, \ldots, z = l$, where each set contains n probes directed at c_0. Note that the number of probes performed is $O(ln^2)$. After collecting these sample points, they are projected onto the (x, y)-plane, and the algorithm of [2] or [3], determine whether there exists an annulus of inner radius r and outer radius R which contains them.

As in Sect. 3, let us define the function

$$f''(n) = f'(n) + \frac{1}{n}\left(((1 + 3\delta)\pi/\alpha)^2 + \pi^2\right)^{\frac{1}{2}} \ . \tag{24}$$

Lemma 5. *Let J be an object of length l with center c_J, and satisfying Assumptions 3, 4, and 5. Let c_0 be any point such that $\operatorname{dist}(c_I, c_0) \leq 2\delta$, and let S be a set of $ln/2\pi$, $n \geq n'_0$, probes directed at c_0 as described above. Then for any point $p \in \operatorname{bd}(J)$ there exists a point $p' \in S$ such that $\operatorname{dist}(p, p') \leq f''(n)$.*

Proof (Sketch). Note that S contains a ring of probes S' such that for all $p' \in S$, $|z(p') - z(p)| \leq \pi/n$. With reference to Fig. 3, one can prove that $\operatorname{dist}(p, p'') \leq f'(n)$ using the same arguments used in the proof of Lemma 2. Using Assumption 5, once can also prove that $\operatorname{dist}(p'', p') \leq \frac{1}{n}\left(((1+3\delta)\pi/\alpha)^2 + \pi^2\right)^{\frac{1}{2}}$. The stated bound then follows from the triangle inequality. □

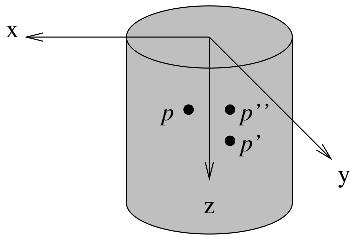


Fig. 3. The proof of Lemma 5.

With this result, and following the arguments of Sect. 3.2, it is not difficult to prove the correctness and running time of the testing procedure for cylinders.

Theorem 4. *There exists a roundness classification procedure for cylinders that can correctly classify any object J of length l and satisfying Assumptions 3, 4, and 5 using $O(l/\operatorname{qual}(J)^2)$ probes and $O(l/\operatorname{qual}(J)^2 \log(l/\operatorname{qual}(J)^2))$ computation time.*

5 Lower Bounds

In this section, we give lower bounds which show that the number of probes used in our testing procedures is optimal up to a constant factor. These lower bounds hold even if the center of the object is known in advance. We begin with a lower bound for planar objects.

Theorem 5. *Any roundness classification procedure that is always correct requires, in the worst case, $\Omega(|1/\operatorname{qual}(I)|)$ probes to classify a planar object I with center $c_I = O$ and satisfying Assumption 1 and Assumption 2.*

Proof (Sketch). Let I and I' be the two objects depicted in Fig. 4. For any $0 \leq \psi \leq \epsilon$, $\operatorname{qual}(I) = -\operatorname{qual}(I') = \psi$. It is not difficult to see that if a procedure uses $o(|1/\psi|)$ probes, then by rotating I' appropriately, it is possible to "hide" the recess in I' so that the procedure cannot differentiate between I and I'. □

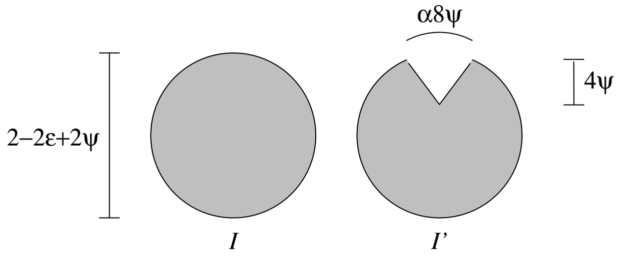


Fig. 4. The proof of Theorem 5.

A similar argument can be used for cylinders. The difference being that the recess in the object J' is a circular cone. That the recess in J' can be hidden comes from the fact that a set of n^2 points in the unit square always contains an empty circle of radius $\Omega(1/n)$.[1]

Theorem 6. *Any roundness classification procedure for cylinders that is always correct requires, in the worst case,* $\Omega(l/\text{qual}(J)^2)$ *probes to classify an object* J *with center* $c_I = O$ *and satisfying Assumptions 3, 4, and 5.*

Acknowledgement

The authors would like to thank Silvia Götz for carefully reading several early revisions of this paper.

References

1. Richard Cole and Chee K. Yap. Shape from probing. *Journal of Algorithms*, 8:19–38, 1987. 129
2. Mark de Berg, Prosenjit Bose, David Bremner, Suneeta Ramaswami, and Gordon Wilfong. Computing constrained minimum-width annuli of point sets. In *Proceedings of the 5th Workshop on Data Structures and Algorithms*, pages 25–36, 1997. 135, 136
3. Christian A. Duncan, Michael T. Goodrich, and Edgar A. Ramos. Efficient approximation and optimization algorithms for computational metrology. In *8th ACM-SIAM Symposium on Data Structures and Algorithms (SODA)*, pages 121–130, 1997. 135, 136
4. Kurt Mehlhorn, Thomas C. Shermer, and Chee K. Yap. A complete roundness classification procedure. In *ACM Symposium on Computational Geometry*, pages 129–138, 1997. 129, 130, 131, 134, 135
5. Chee K. Yap. Exact computational geometry and tolerancing metrology. In David Avis and Jit Bose, editors, *Snapshots of Computational and Discrete Geometry, Vol. 3*. 1994. 130

[1] To see this, divide the square into a $(n+1) \times (n+1)$ grid. Some cells in this grid must have no points in them, therefore an empty circle of diameter $1/(n+1)$ can be placed in one of these empty cells.

Casting with Skewed Ejection Direction*

Hee-Kap Ahn, Siu-Wing Cheng, and Otfried Cheong

Department of Computer Science, HKUST, Hong Kong
{pastel,scheng,otfried}@cs.ust.hk

Abstract. Casting is a manufacturing process in which liquid is poured into a cast (mould) that has a cavity with the shape of the object to be manufactured. The liquid then hardens, after which the cast is removed. We address geometric problems concerning the removal of the cast. A cast consists of two parts, one of which retracts in a given direction carrying the object with it. Afterwards, the object will be ejected from the retracted cast part. In this paper, we give necessary and sufficient conditions to test the feasibility of the cast part retraction and object ejection, where retraction and ejection directions need not be the same. For polyhedral objects, we show that the test can be performed in $O(n^3 \log n)$ time and the cast parts can be constructed within the same time bound. We also give a polynomial time algorithm for finding a feasible pair of retraction and ejection directions for a given polyhedral object.

1 Introduction

The manufacturing industry has at its disposal a wide variety of processes for constructing objects, including gravity casting, injection molding [7,10,15], layered manufacturing (as, for instance, stereolithography [3]), material removal via conventional (or chemical or electrical) machining, deformation (forging, rolling, extrusion, bending), composition (as in composite materials, sintered ceramics, and the like), and spray deposition. In all of these manufacturing contexts, CAD/CAM systems of growing sophistication are presently being introduced. Geometric computing has thus become ubiquitous in the manufacturing industry, and more and more real-world objects begin their life as geometric objects modeled within a computer. The survey by Bose and Toussaint [4,6] gives an overview of these geometric problems and algorithms arising in this area.

The casting process [9,16] consists of two stages. First, liquid is poured into a cavity formed by two cast parts. After the liquid hardens, the movable cast part first retracts from the fixed cast part carrying the object with it. Afterwards, the object is ejected from the retracted cast part. In most existing machinery, the retraction and ejection directions are identical, and previous work on this problem has assumed this restriction on casting. Existing technology for injection molding, however, already has the flexibility to accomodate an ejection direction that is different from the retraction direction of the moving cast part. Exploiting this possibility allows to cast more parts, or to cast parts with simpler moulds, and is the subject of the present paper.

* Research supported in part by RGC CERG HKUST 6074/97E

To simplify our discussion, we will pretend that it is not the part that is ejected from the moving cast part, but that the cast part is removed from the part. In this way, we have symmetry between the two cast parts, and both retraction and ejection are modelled conceptually by removal of a cast part. To simulate the retraction, the fixed cast part will first be removed in a direction opposite to the retraction. Then to simulate the ejection, the remaining cast part will be removed in a direction opposite to the ejection. To summarize, in our model of casting, the two cast parts are to be removed in two given directions and these directions need not be opposite. Note that the ordering of removal is important.

The cast parts should be removed from the object without destroying either cast parts or the object. This ensures that the given object can be mass produced by re-using the same cast parts. The casting process may fail in the removal of the cast parts: if the cast is not designed properly, then one or more of the cast parts may be stuck during the removal phase. The problem we address here concerns this aspect: Given a 3-dimensional object, is there a cast for it whose two parts can be removed after the liquid has solidified? An object for which this is the case is called *castable*.

The 2-dimensional version of castability problem has been studied by Rappaport and Rosenbloom [15]. They presented an $O(n)$ time algorithm to determine whether a simple n-vertex polygon can be decomposed into two monotone chains, which is a sufficient and necessary condition for the polygon to be castable. Hui and Tan [12] gave a heuristic approach to the 3-dimensional problem, assuming opposite directions for cast removal. It is based on testing candidate directions using sample points, and may not find a feasible direction, or may incorrectly return an infeasible direction. Kwong [13] gave the first complete algorithm to determine the feasibility of a given parting direction. He reduced the problem to the hidden surface removal problem in computer graphics by observing that if all the facets can be completely illuminated from the parting direction and its opposite, then the parting direction is feasible. Chen et al. [8] show how to find the parting direction that maximizes the number of completely visible "cavities" in the object. This direction, however, may not be a good parting direction even if one exists. Based on Chen et al.'s work, Hui [11] gave exponential time algorithms that also take cores and inserts into account. Again they are not guaranteed to find a feasible cast. Bose et al. [5] considered a special model of casting, the *sand casting model*, where the partition of the cast into two parts must be done by a plane. Note that even convex polyhedra are not always castable in this model [5].

Finally, Ahn et al. [2] gave, to our knowledge, the first complete algorithm to determine the castability of polyhedral parts for opposite directional cast removal. Given a simple polyhedron with n vertices, they presented an $O(n \log n)$-time algorithm to compute castability in a given direction. They also presented an $O(n^4)$-time algorithm to compute all *combinatorially distinct directions* for which there is a good cast. They showed that there exist polyhedra for which there are $\Omega(n^4)$ combinatorially distinct directions in which there is a good cast.

In this paper we give a complete characterization of castability, under the assumption that the cast has to consist of two parts that are to be removed in two not necessarily opposite directions. We also give an algorithm to verify this condition for polyhedral objects. We do not assume any special separability of the two cast parts, and allow parts of arbitrary genus. The running time of our algorithm for determining the castability of an object with a given pair of directions is $O(n^3 \log n)$.

All the results for opposite cast parts removal in [2,12,13] rely on the property that an object is castable if its boundary surface is completely visible from the two opposite removal directions. This is not true when the removal directions are non-opposite: there are polyhedra whose whole boundary is visible from the removal directions but which are not castable with respect to those directions [2].

For completeness, we also give an $O(n^{15} \log n)$-time algorithm for finding all combinatorially distinct feasible pairs of removal directions. Though the running time is polynomial, the algorithm is clearly of theoretical interest only.

2 Preliminaries

Throughout this paper, $\mathcal{P}$ denotes a polyhedron, that is, a (not necessarily convex) solid bounded by a piecewise linear surface. The union of vertices, edges, and facets on this surface forms the boundary of $\mathcal{P}$, which we denote by $\text{bd}(\mathcal{P})$. We require $\text{bd}(\mathcal{P})$ to be a connected 2-manifold. Each facet of $\mathcal{P}$ is a connected planar polygon, which is allowed to have polygonal holes. Two facets of $\mathcal{P}$ are called *adjacent* if they share an edge. We assume that adjacent facets are not coplanar—they should be merged into one—but we do allow coplanar non-adjacent facets. We also assume that $\mathcal{P}$ is *simple*, which means that no two non-adjacent facets share a point. Our assumptions imply that $\mathcal{P}$ may contain tunnels, but no voids—a polyhedron with a void is not castable anyway.

Our characterization of castability applies to general objects $\mathcal{Q}$. This is important since many industrial parts are not polyhedral. However, our algorithms are only designed for polyhedra.

We call the two removal directions the *red* and the *blue* direction, and we denote them by $\boldsymbol{d}_r$ and $\boldsymbol{d}_b$ respectively. We assume that the outer shape of the cast equals a box denoted by $\mathcal{B}$, which is large enough so that the object is contained strictly in its interior. This assumption is necessary for producing connected cast parts. Our goal is to decompose the cast into two parts which only overlap along their boundaries. The cast part to be removed first is called the *red part* and is denoted by $\mathcal{C}_r$. The other cast part is called the *blue part* and is denoted by $\mathcal{C}_b$. The removal directions for $\mathcal{C}_r$ and $\mathcal{C}_b$ are $\boldsymbol{d}_r$ and $\boldsymbol{d}_b$ respectively. Each of $\mathcal{C}_r$ and $\mathcal{C}_b$ is a connected subset of $\mathcal{B}$. The union of $\mathcal{Q}$ and $\mathcal{C}_r \cup \mathcal{C}_b$ equals $\mathcal{B}$. Note that $\mathcal{B} \setminus (\mathcal{C}_r \cup \mathcal{C}_b)$ is an open set and $\text{cl}(\mathcal{B} \setminus (\mathcal{C}_r \cup \mathcal{C}_b))$ is the object to be manufactured.

3 A Characterization of Castability

We call an object $\mathcal{Q}$ *castable* with respect to $(\boldsymbol{d}_r, \boldsymbol{d}_b)$ if we can translate $\mathcal{C}_r$ to infinity in direction $\boldsymbol{d}_r$ without collision with $\text{int}(\mathcal{Q})$ and $\text{int}(\mathcal{C}_b)$, and then translate $\mathcal{C}_b$ to infinity in direction $\boldsymbol{d}_b$ without collision with $\text{int}(\mathcal{Q})$. The order of removal is important.

We illuminate $\mathcal{Q}$ with two sources of parallel light. The red light source is at infinity in direction $\boldsymbol{d}_r$ and the blue light source is at infinity in direction $\boldsymbol{d}_b$. We say that a point p in space is illuminated by red light if a red ray from direction $\boldsymbol{d}_r$ can reach p without intersecting $\text{int}(\mathcal{Q})$. The definition for a point p being illuminated by blue light is similar. Note that we assume that a light ray will not stop when it grazes the boundary of $\mathcal{Q}$.

There is a (possibly disconnected) subset of $\mathcal{B} \setminus \mathcal{Q}$ not illuminated by red light. We call it the *red shadow volume* and denote it by $\mathcal{V}_r$. Similarly, there is a subset of $\mathcal{B} \setminus \mathcal{Q}$ not illuminated by blue light. We call it the *blue shadow volume* and denote it by $\mathcal{V}_b$. If we sweep $\mathcal{V}_b$ to infinity in direction $\boldsymbol{d}_r$, then we will encounter a set of points in $\mathcal{B}$ and we denote this set of points by $\mathcal{V}_b^*$. Note that $\mathcal{V}_b^*$ includes $\mathcal{V}_b$ itself.

Lemma 1. *If $\mathcal{Q}$ is castable, then $\mathcal{V}_r \subseteq \mathcal{C}_b$ and $\mathcal{V}_b^* \subseteq \mathcal{C}_r$.*

Proof. By definition, $\mathcal{B} \setminus \mathcal{Q}$ is contained in $\mathcal{C}_r \cup \mathcal{C}_b$ and so both $\mathcal{V}_r$ and $\mathcal{V}_b$ are contained in $\mathcal{C}_r \cup \mathcal{C}_b$. Take any point p in $\mathcal{V}_r$. If we move p in direction $\boldsymbol{d}_r$ to infinity, then p will be stopped by $\text{int}(\mathcal{Q})$ as p does not receive any red light. So p cannot be a point in the red cast part $\mathcal{C}_r$. Thus, $\mathcal{V}_r \subseteq \mathcal{C}_b$. By similar analysis, $\mathcal{V}_b \subseteq \mathcal{C}_r$. Since $\mathcal{Q}$ is castable, $\mathcal{C}_r$ can be translated first to infinity in $\boldsymbol{d}_r$ without colliding with $\text{int}(\mathcal{Q})$ and $\text{int}(\mathcal{C}_b)$. Since $\mathcal{V}_b \subseteq \mathcal{C}_r$, we conclude that $\mathcal{V}_b^* \subseteq \mathcal{C}_r$. ⊔⊓

We are now ready to prove the necessary and sufficient condition for an object to be castable.

Theorem 1. *Given an object $\mathcal{Q}$ and removal directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$, $\mathcal{Q}$ is castable if and only if $\mathcal{V}_r$ lies in one connected component of* $\text{cl}(\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q}))$.

Proof. First, we prove that the condition is necessary. Since $\mathcal{Q}$ is castable, $\mathcal{V}_b^* \subseteq \mathcal{C}_r$ and $\mathcal{V}_r \subseteq \mathcal{C}_b$ by Lemma 1. Since $\mathcal{C}_b \subseteq \text{cl}(\mathcal{B} \setminus (\mathcal{C}_r \cup \mathcal{Q}))$, we have $\mathcal{V}_r \subseteq \mathcal{C}_b \subseteq \text{cl}(\mathcal{B} \setminus (\mathcal{C}_r \cup \mathcal{Q})) \subseteq \text{cl}(\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q}))$. Therefore, if $\mathcal{V}_r$ does not lie in one connected component of $\text{cl}(\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q}))$, then so does $\mathcal{C}_b$. This implies that $\mathcal{C}_b$ is not connected, a contradiction.

Second, we prove the sufficiency of the condition. Without loss of generality, let $\boldsymbol{d}_r$ be the positive vertical direction. Let f be the top facet of $\mathcal{B}$. Let M be a thin layer of $\mathcal{B}$ below f and above $\mathcal{Q}$. M exists as $\mathcal{Q}$ lies strictly in the interior of $\mathcal{B}$. Take the connected component R of $\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q})$ that contains $\mathcal{V}_r$. Define the blue cast part $\mathcal{C}_b$ to be $\text{cl}(R \setminus M)$. Then define the red cast part $\mathcal{C}_r$ to be $\text{cl}(\mathcal{B} \setminus (\mathcal{Q} \cup \mathcal{C}_b))$.

We first argue that each cast part is connected. We can ensure that $\mathcal{C}_b$ is connected by adjusting the thickness of M. Since both $\mathcal{C}_r$ and M are completely illuminated by red light, every connected component of $\mathcal{C}_r$ overlaps with M and hence $\mathcal{C}_r$ is connected.

We now show that the cast parts can be removed in order. Since $\mathcal{C}_r$ is completely illuminated by red light, $\mathcal{C}_r$ can be translated to infinity in $\boldsymbol{d}_r$ without colliding with $\text{int}(\mathcal{Q})$. We also claim that this translation of $\mathcal{C}_r$ cannot be obstructed by $\text{int}(\mathcal{C}_b)$. Otherwise, a point p in $\mathcal{C}_r$ can see a point q in $\text{int}(\mathcal{C}_b)$ in direction $\boldsymbol{d}_r$. If the line segment pq contains a point in $\mathcal{V}_b^*$, then q also belongs to $\mathcal{V}_b^*$ which is a subset of $\mathcal{C}_r$ by Lemma 1. Thus, $q \in \mathcal{C}_r$, a contradiction. If the line segment pq does not contain any point in $\mathcal{V}_b^*$, then pq lies $\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q})$. Since $q \in \text{int}(\mathcal{C}_b)$, p also belongs to the connected component of $\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{Q})$ containing $\mathcal{V}_r$. Thus, we also include p in $\mathcal{C}_b$. Since $q \in \text{int}(\mathcal{C}_b)$, we can use the same argument for points in the neighborhood of q to show that every point in the neighborhood of p belongs to $\mathcal{C}_b$. This implies that $p \in \text{int}(\mathcal{C}_b)$ which contradicts that $p \in \mathcal{C}_r$.

After removing $\mathcal{C}_r$, $\mathcal{C}_b$ can be removed to infinity in $\boldsymbol{d}_b$ without colliding with $\text{int}(\mathcal{Q})$ because $\mathcal{C}_b$ does not contain any point in $\mathcal{V}_b$ by definition. ⊔⊓

The condition in Theorem 1 also implies that $\mathcal{V}_r \cap \mathcal{V}_b$ is empty. However, unlike the case where the two removal directions are opposites of each other, the emptiness of $\mathcal{V}_r \cap \mathcal{V}_b$ does not guarantee castability. Figure 1 shows an object that is not castable, even though $\mathcal{V}_r \cap \mathcal{V}_b$ is empty.

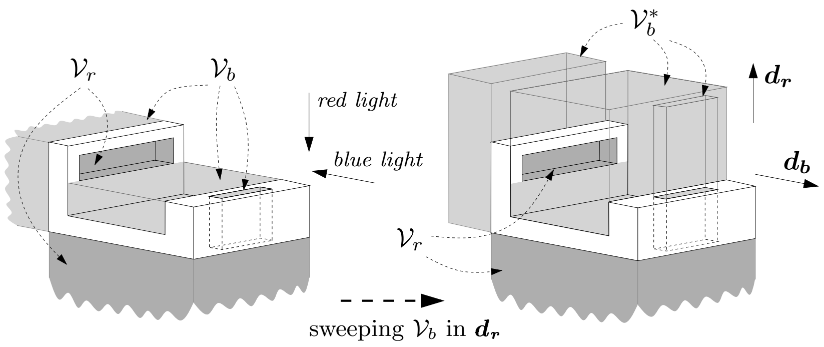


Fig. 1. An object $\mathcal{Q}$ and its shadow volume. $\mathcal{V}_r$ intersects two connected components of $\mathcal{B} \setminus \text{int}(\mathcal{V}_b^* \cup \mathcal{Q})$.

4 Feasibility Test for a Polyhedron

In this section, we present an $O(n^3 \log n)$-time algorithm for testing the feasibility of a pair of removal directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$ for a given polyhedron $\mathcal{P}$. Throughout this section, we treat $\boldsymbol{d}_r$ as the positive vertical direction and we assume that $\mathcal{P}$ lies above the xy-plane. We use $\ell(p)$ to denote the vertical line through a point p.

The *red shadow*, denoted by $\mathcal{S}_r$, is the complement of the set of points on bd($\mathcal{P}$) that can be illuminated by red light from $\boldsymbol{d}_r$ without being obscured by int($\mathcal{P}$). The *blue shadow*, denoted by $\mathcal{S}_b$, is the complement of the set of points on bd($\mathcal{P}$) that can be illuminated by blue light from $\boldsymbol{d}_b$ without being obscured by int($\mathcal{P}$). Their intersection $\mathcal{S}_r \cap \mathcal{S}_b$ is called the *black shadow.*

For each polyhedron edge e, let $h_b(e)$ denote the plane through e and parallel to $\boldsymbol{d}_b$. Then e is a *blue silhouette edge* if it satisfies two requirements. The first requirement is that the two facets incident to e lie in a closed halfspace bounded by $h_b(e)$ and the dihedral angle through int($\mathcal{P}$) is less than π. The second requirement is that if a facet incident to e is parallel to $\boldsymbol{d}_b$, then e should be behind that facet when viewing from direction $\boldsymbol{d}_b$. A *lower blue silhouette edge* is a blue silhouette edge e where $\mathcal{P}$ lies *above* $h_b(e)$ locally at e. Similarly, an *upper blue silhouette edge* is a blue silhouette edge e where $\mathcal{P}$ lies *below* $h_b(e)$ locally at e.

For each lower blue silhouette edge e, imagine that e is a neon tube shooting blue rays in direction $-\boldsymbol{d}_b$. We trace the "sheet" of blue rays emanating from e until they hit int($\mathcal{P}$), or hit an edge or facet parallel to $\boldsymbol{d}_b$ and above int($\mathcal{P}$) locally, or reach infinity in direction $-\boldsymbol{d}_b$. The union of these intercepted or unintercepted blue rays define a subset of the plane $h_b(e)$ called a *lower blue curtain.* Note that a lower blue curtain may pass through a facet of $\mathcal{P}$ parallel to $\boldsymbol{d}_b$. Such a facet must then be locally above $\mathcal{P}$.

For each upper blue silhouette edge e, we define an *upper blue curtain* similarly. We trace the "sheet" of blue rays emanating from e until they hit int($\mathcal{P}$), or hit an edge or facet parallel to $\boldsymbol{d}_b$ and below int($\mathcal{P}$) locally, or reach infinity in direction $-\boldsymbol{d}_b$. The union of these intercepted or unintercepted blue rays form an upper blue curtain. Note that an upper blue curtain may pass through a facet of $\mathcal{P}$ parallel to $\boldsymbol{d}_b$. Such a facet must then be locally below $\mathcal{P}$.

Given a blue silhouette edge e, we use $\Gamma(e)$ to denote the blue curtain defined by e. If $\Gamma(e)$ is nonempty, then it is bounded by a silhouette edge e, edges parallel to $\boldsymbol{d}_b$ called the *side edges*, and a set $\xi(e)$ of polygonal chains opposite to e.

We divide castability testing into three steps. We first verify that the boundary of $\mathcal{P}$ is completely illuminated by red and blue light. That is, $\mathcal{S}_r \cap \mathcal{S}_b$ is empty. Once this test is passed, we then check whether $\mathcal{V}_r \cap \mathcal{V}_b$ is empty. If this test is passed, then we construct the red and blue cast parts and verify that they are connected.

4.1 Testing Emptiness of Black Shadow

The emptiness of $\mathcal{S}_r \cap \mathcal{S}_b$ can be tested in $O(n^2 \log n)$ time as follows. We put a horizontal plane H above $\mathcal{P}$ and compute the projection of $\mathcal{P}$ onto H with the hidden portion removed. The resulting arrangement is known as the visibility map. We project this visibility map vertically downward on the boundary of $\mathcal{P}$. This tells us which part of bd($\mathcal{P}$) is illuminated by red light. An edge in the visibility map is the projection of a polyhedron edge. A vertex in the visibility map is the projection of a polyhedron vertex or the intersection between the projections of two polyhedron edges. Clearly, the size of the visibility map is $O(n^2)$. It can be computed in $O(n^2 \log n)$ time using a plane sweep over the projection of all polyhedron edges to remove the hidden line segments. Output-sensitive algorithms for visibility map computation are also known [1]. Thus, determining the parts of $\mathrm{bd}(\mathcal{P})$ illuminated by red light can be done in $O(n^2 \log n)$. Similarly, we can determine the parts of $\mathrm{bd}(\mathcal{P})$ illuminated by blue light in $O(n^2 \log n)$. We can then decide whether $\mathcal{S}_r \cap \mathcal{S}_b$ is empty by testing the intersection separately on every facet of $\mathrm{bd}(\mathcal{P})$, for instance with a plane sweep algorithm. In total, this test takes time $O(n^2 \log n)$.

4.2 Testing Emptiness of Black Shadow Volume

Once we know that $\mathcal{S}_r \cap \mathcal{S}_b$ is empty, we can determine if the black shadow volume is empty by examining the projections of blue curtains on the xy-plane. This is more efficient than computing $\mathcal{V}_r \cap \mathcal{V}_b$ directly. We show how this is done in the following. Let π be the projection function. We first need a technical lemma. Its proof can be found in the full paper.

Lemma 2. *Suppose that $\mathcal{V}_r \cap \mathcal{V}_b$ is empty. Then for any two blue silhouette edges e and f, $\ell(p) \cap e$ is not above $\ell(p) \cap \Gamma(f)$ for all points $p \in \pi(e) \cap \mathrm{int}(\pi(\Gamma(f)))$.*

The emptiness of the black shadow is now determined by the necessary and sufficient condition stated in the following result.

Lemma 3. *Suppose that $\mathcal{S}_r \cap \mathcal{S}_b$ is empty. Then $\mathcal{V}_r \cap \mathcal{V}_b$ is non-empty if and only if for some lower blue silhouette edge e and blue silhouette edge f, $\ell(p) \cap e$ is not below $\ell(p) \cap \Gamma(f)$ for some point p in $\pi(e) \cap \mathrm{int}(\pi(\Gamma(f)))$ or $\pi(e) \cap \pi(\xi(f))$.*

Proof. We prove sufficiency first. Suppose that p is a point in the intersection of $\pi(e)$ and $\mathrm{int}(\pi(\Gamma(f)))$ such that $\ell(p) \cap e$ is not below $\ell(p) \cap \Gamma(f)$. By Lemma 2, $\ell(p) \cap \Gamma(e)$ is also not above $\ell(p) \cap \Gamma(f)$. Thus, $\ell(p) \cap \Gamma(e) = \ell(p) \cap \Gamma(f)$. We claim that $\Gamma(f)$ is an upper blue curtain, otherwise by definition, the interior of $\Gamma(f)$ would not contain the point $\ell(p) \cap \Gamma(e)$ as e is a lower blue silhouette edge. Now, we can shift $\ell(p)$ as in the proof of Lemma 2 to show that an arbitrary short segment on $\ell(p)$ below $\ell(p) \cap \Gamma(f)$ does not receive red or blue light. Thus $\mathcal{V}_r \cap \mathcal{V}_b$ is nonempty.

The remaining alternative is that $\pi(e)$ touches $\pi(\xi(f))$ at a point p where $\ell(p) \cap e$ is not below $\ell(p) \cap \Gamma(f)$. If we shift $\ell(p)$ slightly in direction $\boldsymbol{d}_b$ to a vertical line ℓ, we claim that ℓ must intersect the interior of a facet σ incident to e. Otherwise, since e is a lower blue silhouette edge, a facet incident to e would face downward and direction $-\boldsymbol{d}_b$ and so this facet would contain a point in black shadow, a contradiction. Observe that ℓ also intersects the interior of $\Gamma(f)$. Whether $\ell \cap \mathrm{int}(\sigma)$ is above or equal to $\ell \cap \Gamma(f)$, we can argue as before to conclude that $\mathcal{V}_r \cap \mathcal{V}_b$ is nonempty.

We now prove necessity. Any facet of $\mathrm{cl}(\mathcal{V}_r \cap \mathcal{V}_b)$ is parallel to $\boldsymbol{d}_b$, or parallel to $\boldsymbol{d}_r$, or on $\mathrm{bd}(\mathcal{P})$. At least one facet σ is parallel to $\boldsymbol{d}_b$, otherwise $\mathrm{cl}(\mathcal{V}_r \cap \mathcal{V}_b)$ would be unbounded. The facet σ cannot receive any red light as it bounds the black shadow volume. Thus, σ must receive some blue light, otherwise σ would be in black shadow which is supposed to be empty. Therefore, σ must lie on some blue curtain $\Gamma(f)$. Let z be a point in $\text{int}(\sigma)$. If we shoot a ray upward from z, the ray hits $\text{bd}(\mathcal{P})$ at a point $v(z)$. Suppose that we move z in the direction $-\boldsymbol{d}_b$. The height of $v(z)$ from $\Gamma(f)$ is monotonically decreasing and remains non-negative. Otherwise, there would be a position such that $v(z)$ becomes a point in the black shadow which is impossible. Before or just when $\ell(z)$ stabs an edge g of $\xi(f)$, $v(z)$ reaches an edge e such that $\mathcal{P}$ lies locally on one side of a vertical plane through e. Otherwise, a facet adjacent to g would contain a point in black shadow, a contradiction. Observe that e is also a lower blue silhouette edge. and $\ell(z) \cap e$ does not lie below $\ell(z) \cap \Gamma(f)$. Clearly, $\pi(e)$ either intersects the interior of $\pi(\Gamma(f))$ or touches $\pi(\xi(f))$ at the intersection of $\ell(z)$ and the xy-plane. ⊔⊓

To test the condition in Lemma 3, we identify all blue silhouette edges, construct the blue curtains, and check the condition in the projection of the blue curtains by a plane sweep. The blue silhouette edges can be identified in $O(n)$ time. To construct its blue curtain we intersect a plane with $\mathcal{P}$ in $O(n \log n)$ time, and so we can construct all blue curtains in time $O(n^2 \log n)$. Consider the condition in Lemma 3. For a lower blue silhouette edge e and a blue silhouette edge f, observe that if $\ell(p) \cap e$ is not below $\ell(p) \cap \Gamma(f)$ for some point p in $\pi(e) \cap \text{int}(\pi(\Gamma(f)))$, then $\ell(q) \cap e$ is not below $\ell(q) \cap \Gamma(f)$ for an endpoint of $\pi(e)$ clipped to $\text{int}(\pi(\Gamma(f)))$. Thus, it suffices to compute the projection of all blue curtains and examine the heights of blue curtains at intersection points in the projection. We will show below that the arrangement formed by the projection of all blue curtains and blue shadow facets has $O(n^3)$ complexity. This implies that we can determine whether $\mathcal{V}_r \cap \mathcal{V}_b$ is empty in time $O(n^3 \log n)$ by computing this arrangement, for instance using a plane sweep, and verifying the heights of blue curtains at all vertices of the arrangement.

It remains to prove the bound on the complexity of the arrangement formed by the projections.

Lemma 4. *Let $\mathcal{P}$ be a polyhedron and let $(\mathbf{d}_r, \mathbf{d}_b)$ be the given pair of removal directions. If $\mathcal{S}_r \cap \mathcal{S}_b$ is empty, then*

(i) For any blue silhouette edges e and f, no edge in $\pi(\xi(e))$ crosses an edge in $\pi(\xi(f))$.
(ii) For any blue silhouette edges e and f, if an edge in $\pi(\xi(e))$ crosses $\pi(f)$ at a point p, then $\ell(p) \cap \xi(e)$ is not above $\ell(p) \cap f$.

Lemma 5. *Let $\mathcal{P}$ be a castable polyhedron. The arrangement formed by the projections of all blue curtains and all blue shadow facets on the xy-plane has complexity $O(n^3)$. This bound is the best possible.*

Proofs can be found in the full paper.

The optimality of the bound follows from the example in Figure 2. The polyhedron shown in Figure 2 (a) has three horizontal upper “legs” and four horizontal lower “legs”, and there is a row of small “teeth”. It is castable using the removal directions shown in Figure 2 (b). Yet every one of the upper "legs" creates a lower blue curtain of linear complexity, and that has a quadratic number of intersections with the curtains formed by the lower "legs". It follows that the arrangement formed by the projections of all the lower curtains has complexity $\Omega(n^3)$, see Figure 2 (c).

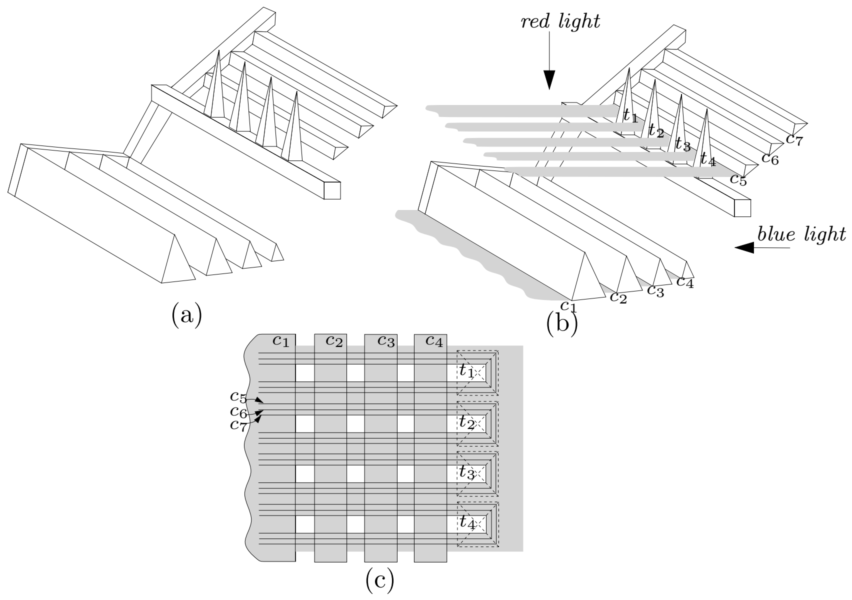


Fig. 2. (a) A polyhedron with 4 vertical pyramids and three upper legs and four lower legs, (b)Some lower blue curtains, and (c) The projection of blue curtains onto the xy-plane.

4.3 Cast Part Construction

Finally, we show how to construct the red and blue cast parts. In the process, we also verify whether the cast parts are connected.

Points on blue shadow facets and points close to and above lower blue curtains are not illuminated by blue light. So they can only be removed in direction $\boldsymbol{d}_r$. Other points encountered while translating these points towards infinity in $\boldsymbol{d}_r$ should then belong to $\mathcal{C}_r$ too. This is exactly the subset of $\mathcal{B}$ swept by the lower envelope of lower blue curtains and blue shadow facets in direction $\boldsymbol{d}_r$. Denote this subset of $\mathcal{B}$ by X. X may be disconnected. Since $\mathcal{P}$ is strictly contained in $\mathcal{B}$, we can take a layer of material M beneath the top facet of $\mathcal{B}$ and above $\mathcal{P}$ and use M to connect all the components in X. By Lemma 5, we can compute the lower envelope of lower blue curtains and blue shadow facets in $O(n^3 \log n)$ time and so $X \cup M$ can then be computed in the same time. $X \cup M$ is our potential red cast part. All the points in $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ are removable in direction $\boldsymbol{d}_b$.

So $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ is our potential blue cast part. However, $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ may be disconnected. Thus, we will try to attach some components in $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ to $X \cup M$ instead.

Such a process is guided by the condition in Theorem 1. Observe that $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ is a subset of $\mathcal{B} \setminus (\mathcal{V}_b^* \cup \mathcal{P})$. From the above analysis, any blue cast part and hence $\mathcal{V}_r$ lies inside $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$. Thus, we can attach every component of $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ not containing any point in $\mathcal{V}_r$ to $X \cup M$. These components are removable in direction $\boldsymbol{d}_r$ as they do not contain points in $\mathcal{V}_r$. In addition, if there are more than one remaining component of $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ containing $\mathcal{V}_r$, then we can abort and report that $\mathcal{P}$ is not castable. Otherwise, we have the cast parts.

It is unnecessary to compute $\mathcal{V}_r$. Every facet bounding $\mathcal{V}_r$ is connected to some red shadow facet. The red shadow facets can be computed in $O(n^2 \log n)$ time using visibility maps. Each red shadow facet lies on a facet bounding $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$. We identify the set of facets bounding $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ that contain the red shadow. Then we test whether this set of facets lie in the same component of $\mathrm{cl}(\mathcal{B} \setminus (X \cup M))$ using a linear-time graph traversal.

Theorem 2. *Let $\mathcal{P}$ be a simple polyhedron with n vertices. Given a pair of directions, we can determine castability and construct cast parts, if castable, of $\mathcal{P}$ in $O(n^3 \log n)$ time and $O(n^3)$ space.*

5 Finding a Pair of Directions

We have seen how to test whether a polyhedron $\mathcal{P}$ is castable in a given pair of directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$. In this section we briefly sketch an algorithm to solve the following problem: Given a polyhedron $\mathcal{P}$, decide whether there is a pair of directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$ in which $\mathcal{P}$ is castable. In fact, we will solve the more general problem of finding all pairs of directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$ for which $\mathcal{P}$ can be cast.

The set of all pairs of directions forms a 4-dimensional parameter space Ψ. We choose an appropriate parameterization that gives rise to algebraic surfaces in Ψ, see for instance Latombe's book [14]. Our goal is to compute that part of Ψ that corresponds to pairs of directions in which $\mathcal{P}$ is castable. As we have proven before, castability depends on a number of simple combinatorial properties: the emptiness of the black shadow, the configuration of the curtain projections, and the connectedness of the blue cast part. We will compute an arrangement of algebraic surfaces in Ψ that includes all pairs of directions where one of these properties could possibly change. The following lemma enumerates all relevant situations. Its proof can be found in the full paper.

Lemma 6. *Let γ_1 and γ_2 be two pairs of directions, such that $\mathcal{P}$ is castable in γ_1 but not in γ_2. Let π be any path in 4-dimensional configuration space Ψ connecting γ_1 and γ_2. Then on π there is a pair of directions $(\boldsymbol{d}_r, \boldsymbol{d}_b)$ such that one of the following conditions holds:*

(i) A facet of $\mathcal{P}$ is parallel to $\boldsymbol{d}_r$ or $\boldsymbol{d}_b$.

(ii) The projection in direction $\boldsymbol{d}_r$ of a vertex v coincides with the projection of an edge e. Here edges and vertices are edges and vertices of $\mathcal{P}$ or of the blue shadow $\mathcal{S}_b$.

(iii) Two polyhedron vertices lie in a plane parallel to the plane determined by $\boldsymbol{d}_r$ and $\boldsymbol{d}_b$.

This characterization can be turned into an algorithm that computes the arrangement of these surfaces and tests each cell separately.

Theorem 3. *Given a polyhedral object $\mathcal{P}$ with nvertices and edges, we can in time $O(n^{15} \log n)$ construct a set of all possible pairs of directions in which $\mathcal{P}$ is castable.*

References

1. Pankaj K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 517–526, 1992. 145
2. H.K. Ahn, M. de Berg, P. Bose, S.W. Cheng, D. Halperin, J. Matoušek, and O. Schwarzkopf. Separating an object from its cast. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997. 140, 141
3. B. Asberg, G. Blanco, P. Bose, J. Garcia, M. Overmars, G. Toussaint, G. Wilfong, and B. Zhu. Feasibility of design in stereolithography. In *Proc. 13th Symp. on FST & TCS*, volume 761 of *Lecture Notes in Computer Science*, pages 228–237. Springer-Verlag, 1993. To appear in Algorithmica. 139
4. P. Bose. *Geometric and Computational Aspects of Manufacturing Processes.* PhD thesis, McGill University, 1994. Also available as UBC Tech Rep 95-02, Dept. of Comp. Sci, Univ. of British Columbia, 1995. 139
5. P. Bose, D. Bremner, and M. van Kreveld. Determining the castability of simple polyhedra. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 123–131, 1994. To appear in Algorithmica. 140
6. P. Bose and G. Toussaint. Geometric and computational aspects of manufacturing processes. *Comput. & Graphics*, 18:487–497, 1994. 139
7. Prosenjit Bose, Marc van Kreveld, and Godfried Toussaint. Filling polyhedral molds. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 210–221. Springer-Verlag, 1993. 139
8. L.L. Chen, S.Y. Chou, and T.C. Woo. Parting directions for mould and die design. *Computer-Aided Design*, 25:762–768, 1993. 140
9. R. Elliot. *Cast Iron Technology.* Butterworths, London, 1988. 139
10. Sándor P. Fekete and Joseph S. B. Mitchell. Geometric aspects of injection molding. Workshop on Geometric and Computational Aspects of Injection Molding, Bellairs Research Institute, February 5–12, 1993. 139
11. K. Hui. Geometric aspects of mouldability of parts. *Computer Aided Design*, 29(3):197–208, 1997. 140
12. K.C. Hui and S.T. Tan. Mould design with sweep operations—a heuristic search approach. *Computer-Aided Design*, 24:81–91, 1992. 140, 141
13. K. K. Kwong. *Computer-aided parting line and parting surface generation in mould design.* PhD thesis, The University of Hong Kong, Hong Kong, 1992. 140, 141
14. J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, 1991. 148

15. A. Rosenbloom and D. Rappaport. Moldable and castable polygons. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 322–327, 1992. 139, 140
16. C.F. Walton and T.J. Opar, editors. *Iron Castings Handbook*. Iron casting society, Inc., 1981. 139

Repairing Flaws in a Picture Based on a Geometric Representation of a Digital Image

Tetsuo Asano[1], Hiro Ito[2], Souichi Kimura[3], and Shigeaki Shimazu[3]

[1] School of Information Science, JAIST, Asahidai, Tatsunokuchi, 923-1292 Japan
[2] Dept. of Information and Computer Sciences, Toyohasi University of Technology Tenpaku-tyo, Toyohasi, 441-8580 Japan
[3] Development Department Graphic Arts Division, Dainippon Screen MFG. Co., Ltd. Teranouchi-agaru 4, Horikawa-dori, Kamigyo-ku, Kyoto, 602-8585 Japan

Abstract. In high-quality image printing it is sometimes required to repair flaws contained in a given image. A simple way for such repair is to paste a flaw region with white and then to move those pixels in the neighborhood by using a tool called an copy-brush. Since it is a very fine operation, it causes great effort to human operators. It is not easy to automate this operation in the existing matrix representation of an image. In our geometric representation of an image as a collection of contour lines for intensity levels this problem is naturally defined as one of reconnecting those contour lines disconnected by a flaw region. An efficient algorithm for reconnecting contour lines is presented based on perfect matching and observations on geometric properties of interconnection paths.

1 Introduction

Recent remarkable development in the printing machine technology is toward more high quality with low costs. In the sense the most important problem is how to automate as many of printing processes based on human operators as possible. In this paper we present an automatic method for one of the tasks, repairing a flaw region in an image, which has been considered to be very hard to be automated. We also demonstrate its effectiveness by experimental results.

In commercial films even a tiny flaw must be removed. Removing electric poles is also the case. An usual way in these cases is to specify a region to be removed as a flaw region and copy pixels from surrounding region by using so-called a "copy-brush" operator. As far as we remain within a framework of matrix representation of images, it is not so easy to automate this operation. The method we propose is based on a different representation of an image, called "contour representation", which represents an image by a collection of contour lines concerning their intensity levels. This is just like a terrain map which is obtained by regarding an intensity level at each pixel as height at the corresponding location. This representation admits geometric (and thus global) treatment of an image. This is a fundamental idea behind our approach.

Assuming the contour representation of an image, the above-stated flaw repairing problem is naturally defined as the following geometric problem. That is, specified a flaw region by a simple polygon and removed all of contour lines included in it, those contour lines intersecting the boundary of the flaw region become disconnected. If we reconnect those disconnected contour lines as natural as possible within the flaw region, the resulting image is expected to look natural. In this paper, we first describe how to reconnect those disconnected contour lines based on perfect matching and observations on geometric properties of interconnection paths.

The authors do not intend to apply the algorithm in this paper to restore random-like texture images. For those random-like images the approach by Hirani and Totsuka [4] based on combined frequency and spatial domain information would be more appropriate.

2 Contour Representation of an Image

A discrete image is usually represented in a matrix form in which each element represents an intensity level of the corresponding pixel (in three matrices for a color image). Contour representation is a different way of representation of an image as a collection of contour lines with intensity levels as heights. A contour line for a level i is the boundary of a region consisting of pixels whose levels are greater than or equal to i.

Our contour representation is different from what is commonly used in computer vision in the sense that contour lines exist between pixels in our method while they pass through the center of pixels in the common one. More concretely, a contour line consists of horizontal and vertical lattice edges between pixels. This difference is important especially for the applications dealt with in this paper in the definition of regions.

3 Flaw Repairing Problem

3.1 Properties of Contour Lines

A flaw region to be removed is specified as a simple polygon F. Then, each contour line intersecting the boundary of F is disconnected by the removal of F. Thus, the problem is how to reconnect those disconnected contour lines "naturally." Although it is not known what is the best way to reconnect those disconnected contour lines so that the resulting image looks natural. Our experience based on experiments suggests that minimization of the total length of chords to be added to reconnect them leads to reasonable results in many cases. Therefore, we settle our goal in this paper to propose an efficient algorithm for connecting contour lines intersecting a flaw region so that the total length of chords to be added in the region for their interconnection is as small as possible.

Intersections of contour lines with the boundary of a flaw region F, called "terminals", are numbered in a clockwise manner along the boundary as $v_0, v_1, \ldots, v_{n-1}$. Since a contour line is directed so that pixels of higher levels lie to its right, there are two types of terminals depending on the directions of their associated contour lines, that is, in one type they enter the flaw region F and in the other type they exit from F. The type of a terminal v_i defined like this is denoted by type(v_i). It is defined by type(v_i) = IN for those terminals at which contour lines enter F and by type(v_i) = OUT otherwise. When the contour line associated with a terminal v_i is the one between levels k and $k+1$, it is denoted by level(v_i) = k (see Fig. 1).

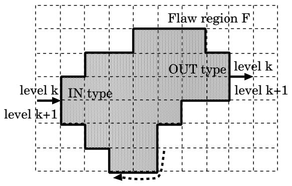


Fig. 1. Classification of contour lines incident to a flaw region by their directions.

No two contour lines pass through the same place in a terrain map unless they are from a cliff. Thus, if we assume that no two contour lines touch each other, the terminals change their associated levels continuously along the boundary of a flaw region. Thus, a terminal of level k is adjacent to those of levels of $k-1, k$ or $k+1$. Here note that contour lines are directed so that higher level pixels lie to their right. This means the following constraints: That is, for a terminal v_i of level k with the type IN (OUT, respectively), its clockwise neighbor is of level $k-1$ ($k+1$, respectively) if it is of type IN (OUT, respectively) and k otherwise (if their types are different).

The above assumption does not hold in practice. It is rather common in a real image that more than two contour lines of different levels pass through the same location. To treat such an image just like a terrain map the following convention suffices: When two pixels of levels j and k ($j < k$) are adjacent on the boundary of F, $(k - j)$ contour lines pass through between them and so we create $k - j$ terminals associated with those levels (although their physical locations are the same). More precisely, if they are IN-type terminals, they are arranged clockwisely along the boundary of F in the decreasing order of their levels, and in the increasing order otherwise. No contradiction is caused by this convention.

We assume the ordering of those terminals along the boundary of F. For two terminals v_i and v_j, a set of terminals encountered when we trace the boundary of F clockwisely from v_i to v_j is denoted by $V_F(v_i, v_j)$. For any terminal v_i on the boundary of F there must be another terminal v_j so that v_i and v_j are interconnected by a contour line because any terminal is originally created by a contour line intersecting F. Such a terminal v_j is called a friend of v_i. Formally, it is defined by

$$v_j \text{ is a friend of } v_i \iff \text{level}(v_j) = \text{level}(v_i) \text{ and type}(v_j) \neq \text{type}(v_i).$$

By the property that intensity levels change continuously along the boundary of F, for any terminal there is a bounded interval along the boundary of F in which its friend exists. We assume level(v_i) > level(v_j) below.

1. type(v_i) = IN and type(v_j) = IN:
   a friend of $v_i \in V_F(v_j, v_i)$, a friend of $v_j \in V_F(v_j, v_i)$.
2. type(v_i) = OUT and type(v_j) = OUT:
   a friend of $v_i \in V_F(v_i, v_j)$, a friend of $v_j \in V_F(v_i, v_j)$.
3. type(v_i) = IN and type(v_j) = OUT:
   a friend of $v_i \in V_F(v_j, v_i)$, a friend of $v_j \in V_F(v_i, v_j)$.
4. type(v_i) = OUT and type(v_j) = IN:
   a friend of $v_i \in V_F(v_i, v_j)$, a friend of $v_j \in V_F(v_j, v_i)$.

Now, we turn to the problem of how to repair a flaw region by interconnecting terminals from disconnected contour lines in a "natural" way. Here we have to note the following constraints (see Fig. 2). When we say that two paths cross each other, it means that one lies in the both sides of the flaw region dissected by the other path.

Constraint 0: Every contour line must become a closed loop without self-intersection (allowing touching itself).
Constraint 1: Two terminals to be interconnected must be of the same level.
Constraint 2: Two terminals to be interconnected must be of different types.
Constraint 3: Terminals must be interconnected only within the flaw region.
Constraint 4: No two contour lines cross each other.

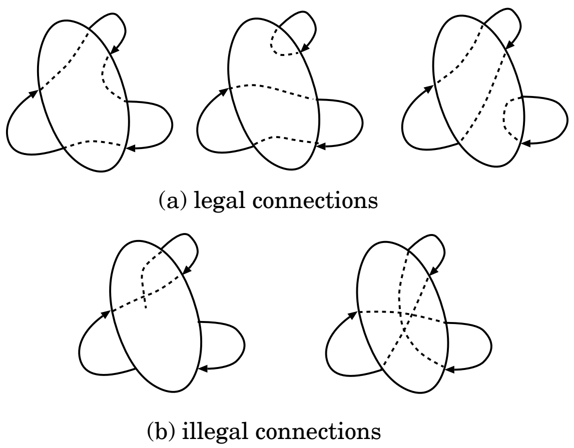


Fig. 2. Constraints on interconnection of disconnected contour lines.

3.2 Algorithm for Reconnecting Disconnected Contour Lines

The problem we consider in this paper is described as follows.

Problem 1. When a region in an image represented by the contour representation is specified as a flaw region by a simple polygon, reconnect those contour lines disconnected by the flaw region so that all of the above-stated constraints are satisfied. In addition, the total length of paths to be added for the reconnection within the flaw region should be minimized.

The previous solution to the problem by the authors was a greedy algorithm in which disconnected contour lines are interconnected one by one in the decreasing order of their associated area within the flaw region. More precisely, after ordering contour lines, based on the dynamic programming they found shortest interconnection paths under the constraints that they do not obstruct connectivity of remaining contour lines. So, it is a level-by-level optimization algorithm and it is questionable whether it obtains globally optimal interconnections.

The algorithm to be proposed in this paper tries to find globally optimal solution to the problem. The basic idea of the algorithm is matching.

First define a graph $G = (V, E)$ as follows: Terminals on the boundary of a flaw region F are *vertices* of the graph and when two terminals v_i and v_j satisfy the constraints 1 and 2, that is, their types are different from each other but their levels are just the same, an *edge* is drawn between corresponding vertices. The *weight* of an edge is a combination of the two *keys*. The *first key* of an edge is the length of a shortest path interconnecting the two corresponding terminals within the flaw region, and is denoted by d_{ij}. Note that the distance is measured by the L_2 metric. The *second key* is the square root of their index difference (modulo n, the number of terminals). Although two different index differences can be considered for two terminals v_i and v_j, i.e., forward (or clockwise) and backward (or counter-clockwise) differences, we take their minimum as their index difference. Formally, the index difference ξ_{ij} between v_i and v_j is defined by

$$\xi_{ij} = \min\{(i - j + n) \bmod n, (j - i + n) \bmod n\}.$$

By using these two keys, we define a weight of an edge (v_i, v_j) by $(d_{ij}, \sqrt{\xi_{ij}})$.

When we compare the weights of two edges, we first compare their first keys and then the second keys if the first keys are the same. The reason will become clear in the following discussions why the first keys are not sufficient.

The graph defined above is bipartite and it is obvious that it has perfect matching since all the terminals can be interconnected by their original contour lines. It is also evident that any matching satisfies all of the above constraints 0 through 3. Thus, the problem of how to prevent crossing between different contour lines and to minimize the total length of contour lines at the same time is left. Here, we show that a perfect matching with minimum weight is a solution to this optimization problem.

Theorem 1. *Interconnecting terminals according to a perfect matching with minimum weight, no two contour lines cross each other.*

Proof. Let v_a, v_b, v_c, and v_d be terminals on the boundary of a flaw region, and assume that a path interconnecting v_a and v_b and that between v_c and v_d cross each other. Then, we want to show that both of the edges (v_a, v_b) and (v_c, v_d) cannot be included in a perfect matching with minimum weight. For this purpose, for any perfect matching M including both of the edges there exists a perfect matching with less weight.

By the definition, we have

$$\text{type}(v_a) \neq \text{type}(v_b),\ \text{type}(v_c) \neq \text{type}(v_d),$$

and

$$\text{level}(v_a) = \text{level}(v_b),\ \text{level}(v_c) = \text{level}(v_d).$$

Before continuing the proof of the theorem, we need some basic observations.

Observation 2 *A shortest path interconnecting two terminals on the boundary of a flaw region F does not intersect itself.*

The observation follows from the fact that any path with self-intersection can be shortcut.

Observation 3 *A shortest path between two terminals on the boundary of a flaw region does not cross another shortest path between another pairs of terminals more than once.*

If they cross each other more than once, a closed region is formed between the first and second intersections p and q. Then, both of the clockwise and counter-clockwise paths between p and q along the boundary of the closed region are shortest paths. It contradicts to the uniqueness of a shortest path in a simple polygon.

Case 1: The four terminals are all of the same level:

By the definition, the shortest path between v_a and v_b crosses the shortest path between v_c and v_d. Let the intersection be p. Then, the arrangement of the four terminals must be either (v_a, v_c, v_b, v_d) or (v_a, v_d, v_b, v_c) clockwisely along the boundary of F. Otherwise, the two shortest paths must cross even number of times, which is impossible.

First consider the case of the arrangement (v_a, v_c, v_b, v_d). Then, the above shortest paths can be decomposed into four parts: the path $\mathrm{P}(v_a, p)$ from v_a to p, the path $\mathrm{P}(p, v_b)$ from p to v_b, the path $\mathrm{P}(v_c, p)$ from v_c to p, and the path $\mathrm{P}(p, v_d)$ from p to v_d. Changing their combinations to $\mathrm{P}(v_a, p) + \mathrm{P}(p, v_d)$ and $\mathrm{P}(v_c, p) + \mathrm{P}(p, v_b)$, the resulting interconnection pattern has different pairs of terminals to be interconnected without increasing the total length of paths. If v_a is of a different type from that of v_c, we should change the combination to $\mathrm{P}(v_a, p) + \mathrm{P}(p, v_c)$ and $\mathrm{P}(p, v_d) + \mathrm{P}(p, v_b)$.

Since $\mathrm{P}(p, v_a) + \mathrm{P}(p, v_d)$ is a path from v_a to v_d, its length is no shorter than a shortest path between them, that is, it is no shorter than d_{ad}. In other words, we have

$$d_{ab} + d_{cd} = |\mathrm{P}(v_a, p)| + |\mathrm{P}(p, v_b)| + |\mathrm{P}(v_c, p)| + |\mathrm{P}(p, v_d)| \geq d_{ad} + d_{cb}$$

if type(v_a) = type(v_c), and

$$d_{ab} + d_{cd} \geq d_{ac} + d_{bd}$$

if type(v_a) $\neq$ type(v_c).

It is possible that equalities hold in the above inequalities. An example is shown in Fig. 3.

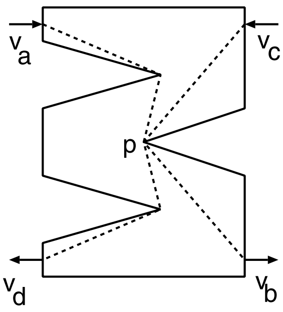


Fig. 3. The case in which the sum of the shortest path lengths cannot be decreased.

On the other hand, taking the arrangement of terminals into accounts, we have the following equalities concerning the index difference. When the index difference between v_a and v_b is given by their backward difference and that between v_c and v_d is given by their forward difference, we have

$\xi_{ab} = \xi_{ad} + \xi_{db}$ and $\xi_{cd} = \xi_{cb} + \xi_{bd}$.

Thus, based on a simple observation that when $0 < p < q < r < s$ and $p + s \leq q + r$, we have $\sqrt{p} + \sqrt{s} < \sqrt{q} + \sqrt{r}$, the following inequalities follow.

$\sqrt{\xi_{ab}} + \sqrt{\xi_{cd}} > \sqrt{\xi_{ad}} + \sqrt{\xi_{bc}}$, and

$\sqrt{\xi_{ab}} + \sqrt{\xi_{cd}} = \sqrt{\xi_{ad} + \xi_{bd}} + \sqrt{\xi_{bc} + \xi_{bd}} > \sqrt{\xi_{ad} + \xi_{bd} + \xi_{bc}} + \sqrt{\xi_{bc}} \geq \sqrt{\xi_{ac}} + \sqrt{\xi_{bd}}$.

The proof for the case in which the index difference between v_a and v_b is given by their backward difference and that between v_c and v_d is given by the forward one is similar to that for the above case.

Difficulty arises when both of the indices are given in the same direction. For example, let us suppose that both of them are given by the backward indices. In this case, we have

$\xi_{ab} = (a - b + n) \bmod n$,

$\xi_{cd} = (c - d + n) \bmod n$

and thus we have

$\xi_{ab} = \xi_{ad} + \xi_{db}$ and $\xi_{cd} = \xi_{ca} + \xi_{ad}$.

Therefore, if the type of v_a is different from that of v_c, interconnecting v_a with v_c and v_b with v_d results in smaller sum of their index differences and the sum of the square roots of the index differences also decreases.

On the other hand, if v_a and v_c are of the same type, it is not so obvious. In this case we are going to interconnect v_a with v_d and v_b with v_c. Then, concerning their index differences, we have the equality

$\xi_{ab} + \xi_{cd} \leq \xi_{ad} + \xi_{bc}$.

However, the equality implies

$\xi_{ad} < \min\{\xi_{ab}, \xi_{cd}\}$.

By the simple observation above we have

$\sqrt{\xi_{ab}} + \sqrt{\xi_{cd}} > \sqrt{\xi_{ad}} + \sqrt{\xi_{bc}}$.

Based on the observations above, it is seen that the combination (v_a, v_d) and (v_b, v_c) has smaller weight in the case of $\text{type}(v_a) = \text{type}(v_c)$, and that of (v_a, v_c) and (v_b, v_d) has smaller weight than that for (v_a, v_b) and (v_c, v_d) if $\text{type}(v_a) \neq \text{type}(v_c)$.

If the terminals are arranged in the order v_a, v_c, v_b, and v_d, we have similar discussion by reversing the directions of index differences.

The proof proceeds similarly for the remaining cases.

Based on the theorem, we can find an optimal interconnection pattern by the following algorithm. First, for each intensity level we enumerate all the crossings between contour lines of that level and the boundary of a flaw region, and after classifying those crossings into IN-type and OUT-type, for each pair of crossings of different types we calculate the length of a shortest path interconnecting them within the flaw region and their index difference. These informations are summarized as a bipartite graph. Since a flaw region is specified as a simple polygon, if we partition its inside into triangles, for each terminal we can compute the shortest path tree from the terminal (and the geodesic distances from the terminal to other terminals) in time proportional to the number of vertices of the polygon [3]. Thus, if we have n crossings and the flaw region is specified as an m-gon, we can build such a bipartite graph in time $O(n^2 + mn)$. Then, we find a minimum-weight matching for this bipartite graph. Some known algorithms are available for this purpose [2]. The best known algorithm runs in $O(n^{2.5})$ time, which leads to the following theorem:

Theorem 4. *Let n be the number of contour lines which cross a flaw region. Then, we can find an optimal interconnection pattern for contour lines disconnected by a flaw region in time $O(n^{2.5} + nm)$, where m is the number of vertices of a polygon which forms a flaw region.*

4 Experimental Results

We have implemented our algorithm against several image data. Examples we have tested are to remove electric poles from a street, to remove claw's feet around eyes, and removing characters in a wine bottle without eliminating highlight part, and so on. The results were satisfactory in all those cases. One of the examples is shown in Fig. 4 in which the flaw region depicted by a white region is restored in the resulting image below. A similar result could be achieved by manual operations using so-called "copy-brush" which copy pixels in the surrounding region. A manual operation, however, takes time and cost, and the quality of the resulting picture depends heavily on experience of human operators. This is why automatic processing is required.

5 Conclusions

In this paper we have presented an algorithm for removing flaws in a picture and described experimental results. The key idea to the success is the use of perfect matching for global optimum. One important contribution of this paper is that the new algorithm can guarantee global optimal solutions in less time than the previous one. That is, our previous algorithm found an optimal solution for each level, but it does not guarantee an optimal solution for all levels.

The contour representation of an image as a collection of contour lines is a new representation proposed by the authors. It allows geometric treatment of an image, which is especially important for getting global view. This is another key to the success. One drawback of the representation is that it takes much storage. In fact, the total length of the contour lines may be much larger than the total number of pixels (or image size). In our application, however, we do not need to convert the whole image into its corresponding contour line representation. We are just required to have contour representation for the region specified as a flaw region. Thus, the overhead due to the representation is not so much. Conversion from contour representation to the conventional matrix form is straightforward.

Acknowledgment

The authors would like to their thanks to Kazuhiro Nakai for his valuable supports in our implementation. This research is partly supported by the Grant-in-Aid of Ministry of Science, Culture and Education of Japan.

References

1. T. Asano and S. Kimura: “Contour Representation of an Image with Applications,” Proc. SPIE’s International Symposium on Vision Geometry IV, pp.14-22, San Diego, July 1995.
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest: “Introduction to Algorithms,” The MIT Press, 1990. 156
3. L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan: “Linear time algorithm for visibility and shortest path problems inside simple polygons,” in Proc. 2nd ACM Symp. on Computational Geometry, pp.1-13, 1986. 156
4. A. N. Hirani and T. Totsuka: “Combining Frequency and Spatial Domain Information for Fast Interactive Image Noise Removal,” Proc. SIGGRAPH’96, pp.269-276, 1996. 150

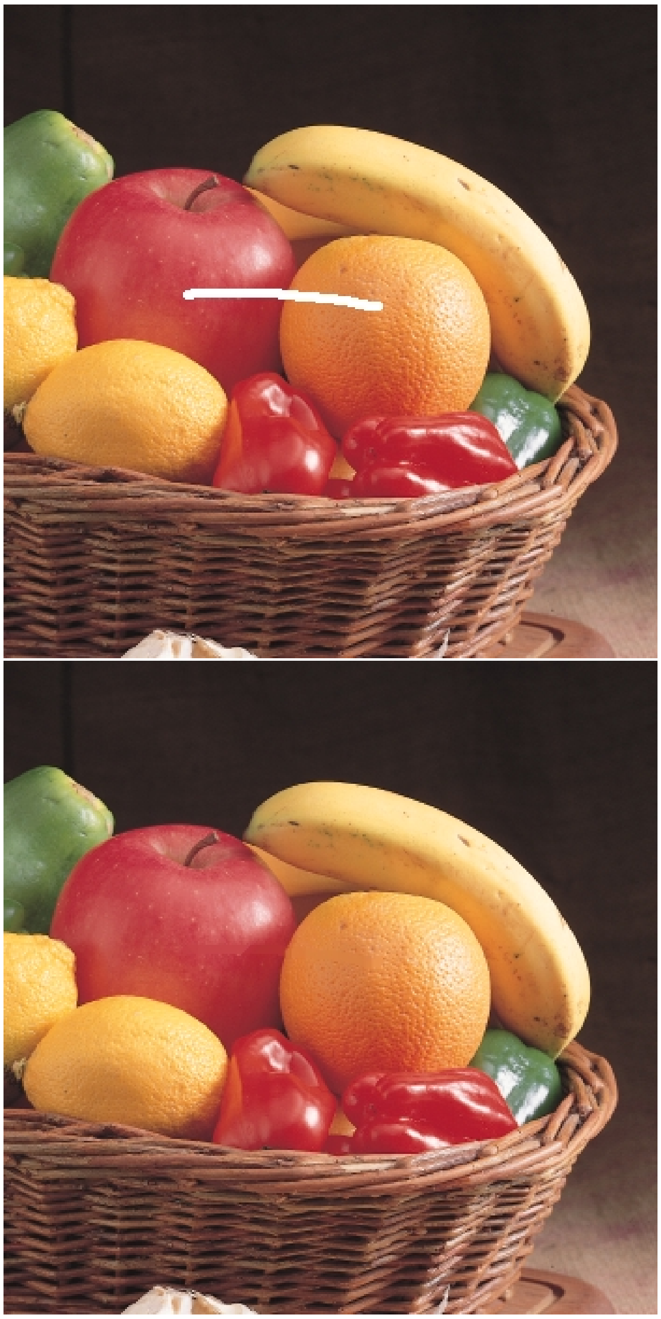


Fig. 4. An experimental result. A picture with a flaw region above and the restored picture image below.

k-Edge and 3-Vertex Connectivity Augmentation in an Arbitrary Multigraph[⋆]

Toshimasa Ishii, Hiroshi Nagamochi, and Toshihide Ibaraki

Department of Applied Mathematics and Physics
Kyoto University, Kyoto 606-8501, Japan
{ishii,naga,ibaraki}@kuamp.kyoto-u.ac.jp

Abstract. Given an undirected multigraph $G = (V, E)$ and two positive integers ℓ and k, the edge-and-vertex connectivity augmentation problem asks to augment G by the smallest number of new edges so that the resulting multigraph becomes ℓ-edge-connected and k-vertex-connected. In this paper, we show that the problem with a fixed ℓ and $k = 3$ can be solved in polynomial time for an arbitrary multigraph G.

1 Introduction

The problem of augmenting a graph by adding the smallest number of new edges to meet edge-connectivity or vertex-connectivity requirement has been extensively studied as an important subject in network design, and many efficient algorithms have been developed so far. However, it was only very recent to have algorithms for augmenting both edge-connectivity and vertex-connectivity simultaneously (see [6,7,8] for those results).

Let $G = (V, E)$ stand for an undirected multigraph with a set V of vertices and a set E of edges. We denote the number of vertices by n, and the number of pairs of adjacent vertices by m. The local edge-connectivity $\lambda_G(x, y)$ (resp., the local vertex-connectivity $\kappa_G(x, y)$) is defined to be the maximum ℓ (resp., k) such that there are ℓ edge disjoint (resp., k internally vertex disjoint) paths between x and y in G (where at most one edge between x and y is allowed in the set of internally vertex disjoint paths). The *edge-connectivity* and *vertex-connectivity* of G are defined by $\lambda(G) = \min\{\lambda_G(x, y) \mid x, y \in V, x \neq y\}$ and $\kappa(G) = \min\{\kappa_G(x, y) \mid x, y \in V, x \neq y\}$. Let r be a function: $\binom{V}{2} \rightarrow Z^+$, where $\binom{V}{2}$ denotes the set of unordered pairs of $x, y \in V$ and Z^+ denotes the set of nonnegative integers. We call a multigraph G r_λ-*edge-connected* if $\lambda_G(x, y) \geq r_\lambda(x, y)$ for all $x, y \in V$. Analogously, G is called r_κ-*vertex-connected* if $\kappa_G(x, y) \geq r_\kappa(x, y)$ for all $x, y \in V$. The *edge-connectivity augmentation problem* (resp., the *vertex-connectivity augmentation problem*) asks to augment G by the smallest number of new edges so that the resulting multigraph G' becomes r_λ-edge-connected (resp., r_κ-vertex-connected).

⋆ This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan, and the subsidy from the Inamori Foundation.

As to the edge-connectivity augmentation problem, Watanabe and Nakamura [10] first proved that the problem with $r_\lambda(x,y) = \ell$ for all $x, y \in V$ can be solved in polynomial time for any given integer ℓ. For a general requirement function r_λ, Frank [2] showed by using Mader's edge-splitting theorem that the problem can be solved in polynomial time.

As to the vertex-connectivity augmentation problem, the problem of making a $(k-1)$-vertex-connected multigraph k-vertex-connected was shown to be polynomially solvable for $k = 2$ [1] and for $k = 3$ [11]. It was later found out that, for $k \in \{2, 3, 4\}$, the vertex-connectivity augmentation problem can be solved in polynomial time in [1,4] (for $k = 2$), [3,11] (for $k = 3$), and [5] (for $k = 4$), even if the input multigraph G is not necessarily $(k-1)$-vertex-connected. However, whether there is a polynomial time algorithm for an arbitrary constant k was still an open question (even if G is $(k-1)$-vertex-connected).

Hsu and Kao [6] first treated the problem of augmenting edge-connectivity and vertex-connectivity simultaneously, and presented a linear time algorithm for augmenting $G = (V, E)$ with two specified subsets X and Y of V by adding the smallest number of edges so that the resulting multigraph G' satisfies $\lambda_{G'}(x, x') \geq 2$ for all $x, x' \in X$ and $\kappa_{G'}(y, y') \geq 2$ for all $y, y' \in Y$. The connectivity augmentation problem in the general setting was first studied in [7,8]. For two given functions $r_\lambda, r_\kappa : \binom{V}{2} \to Z^+$, we say that G is (r_λ, r_κ)-*connected* if G is r_λ-edge-connected and r_κ-vertex-connected. The *edge-and-vertex-connectivity augmentation problem*, denoted by EVAP(r_λ, r_κ), asks to augment G by adding the smallest number of new edges so that the resulting multigraph G' becomes (r_λ, r_κ)-connected, where $r_\lambda(x,y) \geq r_\kappa(x,y)$ is assumed for all $x, y \in V$ without loss of generality. When a requirement function r_κ satisfies $r_\kappa(x,y) = k \in Z^+$ for all $x, y \in V$, this problem is also denoted as EVAP(r_λ, k). The authors presented algorithms EV-AUGMENT [7] and EV-AUGMENT3 [8]. The first algorithm solves EVAP$(r_\lambda, 2)$ in $O(n^3 m \log(n^2/m))$ time, and the second solves EVAP$(\ell, 3)$ in $O(n^4)$ time, under the assumption that ℓ is a fixed constant and the input multigraph is already 2-vertex-connected. However, it is left open whether EVAP$(\ell, 3)$ with a fixed ℓ can be solved in polynomial time for an arbitrary input multigraph G.

In this paper, we consider EVAP$(\ell, 3)$ for an arbitrary input multigraph $G = (V, E)$, which is not necessarily 2-vertex-connected. It seems difficult to extend directly the above EV-AUGMENT3 to this case, since many properties used in EV-AUGMENT3 heavily depend on the 2-vertex-connectivity of the input multigraph. Alternatively, one may first apply EV-AUGMENT to an input multigraph G to obtain a $(\ell, 2)$-connected multigraph G', and then apply EV-AUGMENT3 to G' to obtain a $(\ell, 3)$-connected multigraph G''. However, in this case, the resulting multigraph G'' may not be optimally augmented from the original multigraph G.

In this paper, we first derive two lower bounds $\lceil \alpha(G)/2 \rceil$ and $\beta(G)$ on $opt(G)$, where $opt(G)$ is the optimal value of EVAP$(\ell, 3)$. We next obtain a $(\ell, 2)$-connected multigraph $G_2 = (V, E \cup F)$ with $|F| = \lceil \alpha(G)/2 \rceil$. We show that such G_2 can be computed by applying EV-AUGMENT. We then apply several procedures used in EV-AUGMENT3 in order to replace some edges in F with the same number of edges to attain the 3-vertex-connectivity while preserving its $(\ell, 2)$-connectivity. However, those procedures cannot be directly used unless G is 2-vertex-connected. To remedy this, we show that there exist some edges in F for which the procedures can be applied. As a result of applying these procedures, we can show that either $opt(G) = \max\{\lceil \alpha(G)/2 \rceil, \beta(G)\}$ or $opt(G) \leq 2\ell - 3$ holds and that a set of the smallest number of new edges can then be constructed in $O((2\ell - 3)n^{4\ell-3} + n^2m + n^3 \log n)$ time. Furthermore, we can show that if $\delta(G) \geq \ell$ or $opt(G) \geq 2\ell - 2$ holds, then it can be found in $O(n^2m + n^3 \log n)$ time, and if $opt(G) \leq 2\ell - 3$ holds, then a feasible solution F' with $|F'| \leq \min\{2opt(G) - 1, 2\ell - 3, opt(G) + (\ell + 1)/2\}$, can be found in $O(n^2m + n^3 \log n)$ time. The entire algorithm is called EV-AUGMENT3*.

In Section 2, after introducing basic definitions and the concept of edge-splitting, we derive a lower bound on the optimal value of EVAP$(\ell, 3)$ for an arbitrary multigraph G. In Sections 3, 4 and 5, we give an outline of EV-AUGMENT3*.

2 Preliminaries

2.1 Definitions

For a multigraph $G = (V, E)$, an edge with end vertices u and v is denoted by (u, v). Given two disjoint subsets of vertices $X,\ Y \subset V$, we denote by $E_G(X, Y)$ the set of edges connecting a vertex in X and a vertex in Y, and denote $c_G(X, Y) = |E_G(X, Y)|$. A singleton set $\{x\}$ is also denoted x. In particular, $E_G(u, v)$ is the set of multiple edges with end vertices u and v and $c_G(u, v) = |E_G(u, v)|$ denotes its multiplicity. For a subset $V' \subseteq V$ (resp., $E' \subseteq E$) in G, $G[V']$ (resp., $G[E']$) denotes the subgraph induced by V' (resp., E'), and we denote $G[V - V']$ (resp., $G[E - E']$) simply by $G - V'$ (resp., $G - E'$). For an edge set F, we denote by $V[F]$ the set of end vertices of edges in F. If F satisfies $F \cap E = \emptyset$, we denote $G = (V, E \cup F)$ by $G + F$. A *partition* $X_1, \cdots, X_t$ of a vertex set V is a family of nonempty disjoint subsets X_i of V whose union is V, and a *subpartition* of V is a partition of a subset of V. A *cut* is defined to be a subset X of V with $\emptyset \neq X \neq V$, and the *size* of a cut X is defined by $c_G(X, V - X)$, which may also be written as $c_G(X)$. In particular, $c_G(v)$ for $v \in V$ denotes the *degree* of v. Let $\delta(G)$ denote the minimum degree of G. We say that a cut X *intersects* another cut Y if none of subsets $X \cap Y$, $X - Y$ and $Y - X$ is empty, and X *crosses* Y if in addition $V - (X \cup Y) \neq \emptyset$. A family $\mathcal{X}$ of subsets $X_1, \cdots, X_u$ is called *laminar* if no two subsets in $\mathcal{X}$ intersect each other. A multigraph G is called ℓ-edge-connected if $\lambda(G) \geq \ell$. For a subset X of V, a vertex $v \in V - X$ is called a *neighbor* of X if it is adjacent to some vertex $u \in X$, and the set of all neighbors of X is denoted by $\Gamma_G(X)$. A maximal connected subgraph G' in a multigraph G is called a *component* of G, and the number of components in G is denoted by $p(G)$. A *disconnecting set* of G is defined as a cut S of V such that $p(G - S) > p(G)$ holds and no $S' \subset S$ has this property. Let $\bar{G}$ denote the simple graph obtained from G by replacing multiple edges in $E_G(u, v)$ by a single edge (u, v) for all $u, v \in V$. A component G' of G with $|V(G')| \geq 3$ always has a disconnecting set unless $\bar{G}'$ is a complete graph. If G is connected and contains a disconnecting set, then a disconnecting set of the minimum size is called a *minimum disconnecting set*, whose size is equal to $\kappa(G)$. A cut $T \subset V$ is called *tight* if $\Gamma_G(T)$ is a minimum disconnecting set in G. A tight set T is called *minimal* if no proper subset T' of T is tight (hence, the induced subgraph $G[T]$ is connected). A disconnecting set S is called a *disconnecting vertex* (resp., *disconnecting pair*) if $|S| = 1$ (resp., $|S| = 2$). We say that a disconnecting set $S \subset V$ *disconnects* two disjoint subsets Y and Y' of $V - S$ if no two vertices $x \in Y$ and $y \in Y'$ are connected in $G - S$. For a disconnecting set S, there is a unique component X of G such that $X \supseteq S$, and we call the components in $G[X] - S$ *the S-components.*

2.2 Edge-Splitting

Given a multigraph $H = (V \cup \{s\}, E)$, a designated vertex s, vertices $u, v \in \Gamma_H(s)$ (possibly $u = v$) and a nonnegative integer $\delta \leq \min\{c_H(s, u), c_H(s, v)\}$, we construct multigraph $H' = (V \cup \{s\}, E')$ from H by deleting δ edges from $E_H(s, u)$ and $E_H(s, v)$, respectively, and adding new δ edges to $E_H(u, v)$. We say that H' is obtained from H by *splitting* δ pair of edges (s, u) and (s, v). A sequence of splittings is *complete* if the resulting multigraph H' has no neighbor of s. The following theorem is due to Lovász [9].

Theorem 1. *Let $H = (V \cup \{s\}, E)$ be a multigraph with a designated vertex s and an integer $\ell \geq 2$ such that $c_H(s)$ is an even integer and $\lambda_H(x, y) \geq \ell$ for all pairs $x, y \in V$. Then, for any neighbor u of s, there is a neighbor v (possibly $v = u$) such that $\lambda_{H'}(x, y) \geq \ell$ for all $x, y \in V - s$ in the multigraph H' resulting from H by splitting one pair of edges (s, u) and (s, v).* □

By applying this repeatedly, we see that there always exists a complete splitting at s such that the resulting multigraph H' satisfies $\Gamma_{H'}(s) = \emptyset$ and $\lambda_{H'}(x, y) \geq \ell$ for all $x, y \in V$.

2.3 Lower Bound on the Number of New Edges

In the subsequent discussion, we consider EVAP$(\ell, 3)$ for an arbitrary multigraph G, and assume $\ell \geq 4$ (since the problem is equivalent to the 3-vertex-connectivity augmentation problem if $\ell = 3$). In this section, we derive two types of lower bounds $\alpha(G)$ and $\beta(G)$ on the optimal value *opt*(G) to EVAP$(\ell$,3).

Let X be a cut in G. To make G $(\ell, 3)$-connected, it is necessary to add at least $\max\{\ell - c_G(X), 0\}$ edges between X and $V - X$, or at least $\max\{3 - |\Gamma_G(X)|, 0\}$ edges between X and $V - X - \Gamma_G(X)$ if $V - X - \Gamma_G(X) \neq \emptyset$. Given a subpartition $\mathcal{X} = \{X_1, \cdots, X_{q_1}, X_{q_1+1}, \cdots, X_{q_2}\}$ of V, where $V - X_i - \Gamma_G(X_i) \neq \emptyset$ holds for $i = q_1 + 1, \cdots, q_2$, we can sum up "deficiency" $\max\{\ell - c_G(X_i), 0\}$, $i = 1, \cdots, q_1$, and $\max\{3 - |\Gamma_G(X_i)|, 0\}$, $i = q_1 + 1, \cdots, q_2$. Adding one edge to G contributes to the deficiency of at most two cuts in $\mathcal{X}$. Hence, to make G $(\ell, 3)$-connected, we need at least $\lceil \alpha(G)/2 \rceil$ new edges, where

$$\alpha(G) = \max_{\text{all subpartitions } \mathcal{X}} \left\{ \sum_{i=1}^{q_1} (\ell - c_G(X_i)) + \sum_{i=q_1+1}^{q_2} (3 - |\Gamma_G(X_i)|) \right\}, \quad (2.1)$$

and the maximum is taken over all subpartitions $\mathcal{X} = \{X_1, \cdots, X_{q_1}, X_{q_1+1}, \cdots, X_{q_2}\}$ of V with $V - X_i - \Gamma_G(X_i) \neq \emptyset$, $i = q_1 + 1, \cdots, q_2$.

We now consider another case in which different type of new edges become necessary. For a pair of two vertices $S = \{v, v'\}$ of G, let $T_1, \cdots, T_r$ denote all the components in $G - S$, where $r = p(G - S)$ (note that S may not be a disconnecting pair in G). To make G 3-vertex-connected, a new edge set F must be added to G so that all T_i form a single connected component in $(G + F) - S$. For this, it is necessary to add

(i) at least $p(G - S) - 1$ edges to connect all components in $G - S$.

Moreover, if $\ell > c_G(u)$ holds for a $u \in S$, then it is necessary to add at least $\ell - c_G(u)$ edges in order to make G ℓ-edge-connected. Since adding an edge between v and v' contribute to the requirement of both v and v', we require

(ii) at least $\max\{\ell - c_G(v), \ell - c_G(v'), 0\}$ edges.

In the above, no edge in (i) is incident to v or v', while all edges in (ii) are incident to v or v'; hence there is no edge that belongs to both (i) and (ii). Therefore, it is necessary to add

(iii) at least $p(G - S) - 1 + \max\{\ell - c_G(v), \ell - c_G(v'), 0\}$ edges for $S = \{v, v'\}$.

This means that the following number of new edges are necessary to make G $(\ell, 3)$-connected.

$$\beta(G) = \max_{\substack{\text{all vertex pairs} \\ S = \{v, v'\} \text{ in } G}} \left[p(G - S) - 1 + \max\{\ell - c_G(v), \ell - c_G(v'), 0\} \right]. \quad (2.2)$$

Lemma 1. (Lower Bound) *$\gamma(G) \leq opt(G)$, where $\gamma(G) = \max\{\lceil \alpha(G)/2 \rceil, \beta(G)\}$.* □

Based on this, we shall prove the next result in this paper.

Theorem 2. *Let G be an arbitrary multigraph with n vertices and m adjacent vertex pairs.*
(1) For any integer $\ell \geq 4$, $\gamma(G) \leq opt(G) \leq \max\{\gamma(G), 2\ell - 3\}$ holds and an optimal solution of EVAP$(\ell, 3)$ *can be found in $O((2\ell - 3)n^{4\ell - 3} + n^2 m + n^3 \log n)$ time.*
(2) If $\delta(G) \geq \ell$ or $\gamma(G) \geq 2\ell - 2$, then $opt(G) = \gamma(G)$ holds and an optimal solution F can be found in $O(n^2 m + n^3 \log n)$ time.
(3) If $\gamma(G) \leq 2\ell - 3$, then a feasible solution F' of EVAP$(\ell$,3) *such that $|F'| \leq \min\{2opt(G) - 1, 2\ell - 3, opt(G) + (\ell + 1)/2\}$, can be found in $O(n^2 m + n^3 \log n)$ time.* □

3 Algorithm for EVAP(ℓ,3)

Given a multigraph $G = (V, E)$, let $\mathcal{P}_3(G)$ denote the set of unordered pairs $\{x, y\}$ of vertices $x, y \in V$ with $\kappa_G(x, y) \geq 3$. Thus $\mathcal{P}_3(G) = \binom{V}{2}$ if $\kappa(G) \geq 3$. For a subset $F \subseteq E$ in G, an operation of removing a subset F' from F followed by adding a set F'' of new edges with $|F''| = |F'|$ to F is called a *shifting* in F, and denoted by F''/F'. In particular, a shifting F''/F' in F is called a *switching* if it does not change the degree $c_G(v)$ of any vertex v in G. Given an ℓ-edge-connected multigraph G, a sequence of switchings or shiftings of edges is called *feasible* to G if the resulting multigraph G' remains ℓ-edge-connected and $\mathcal{P}_3(G') \supseteq \mathcal{P}_3(G)$ holds.

We now present a polynomial time algorithm EV-AUGMENT3* for solving EVAP(ℓ,3). The proofs of the properties and the analysis of the time complexity are omitted due to space limitation. An example of computational process of EV-AUGMENT3* is shown in Fig. 1.

Algorithm EV-AUGMENT3*
Input: An undirected multigraph $G = (V, E)$ ($|V| \geq 4$) and an integer $\ell \geq 4$.
Output: A set of new edges F with $|F| = opt(G)$ such that $G + F$ is $(\ell, 3)$-connected.
Step I (Addition of vertex s and associated edges): Add a new vertex s together with a set F_1 of new edges between s and V such that the resulting $G_1 = (V \cup \{s\}, E \cup F_1)$ satisfies

$$\begin{aligned} c_{G_1}(X) \geq \ell \quad & \text{for all cuts } X \subset V, && (3.1)\\ |\Gamma_G(X)| + |\Gamma_{G_1}(s) \cap X| \geq 3 \quad & \text{for all cuts } X \subset V \text{ with } V - X - \Gamma_G(X) \neq \emptyset \\ & \text{and } |\Gamma_G(X)| + |X| \geq 3, && (3.2)\\ |\Gamma_G(x)| + c_{G_1}(s, x) \geq 3 \quad & \text{for all } x \in V, && (3.3) \end{aligned}$$

and $|F_1|$ is *minimum* subject to (3.1) – (3.3). We describe in Section 4 how to find such F_1.

Property 1. $|F_1| = \alpha(G)$. □

If $c_{G_1}(s)$ is odd, then we add to F_1 a new edge $\hat{e} = (s, w)$ for an arbitrary vertex $w \in V$ so that $c_{G_1}(s)$ becomes even. Call this edge $\hat{e} = (s, w)$ *extra*.
Step II (Edge-splitting): Here we show the following property which is stronger than Theorem 1.

Property 2. There is a complete splitting at s in G_1 such that the resulting multigraph $G_2 = (V, E \cup F_2)$ is $(\ell, 2)$-connected. □

Now if $\kappa(G_2) \geq 3$ holds, then we are done, because $|F_2| = |F_1|/2 = \lceil \alpha(G)/2 \rceil$ attains the lower bound of Lemma 1. Otherwise ($\kappa(G_2) = 2$), we can observe from (3.2) and (3.3) that the following holds for the family of minimal tight sets in G_2, denoted by $\mathcal{T}(G_2)$.

$$\begin{aligned}&T \cap V[F_2] \neq \emptyset \text{ for all } T \in \mathcal{T}(G_2),\\ &|E_{G_2}(x) \cap F_2| \geq 2 \text{ for all } T = \{x\} \in \mathcal{T}(G_2) \text{ with } |\Gamma_G(x)| = 1,\\ &|E_{G_2}(x) \cap F_2| \geq 3 \text{ for all } T = \{x\} \in \mathcal{T}(G_2) \text{ with } |\Gamma_G(x)| = 0.\end{aligned} \tag{3.4}$$

Step III (Switching edges): Now the current $G_2 = (V, E \cup F_2)$ is $(\ell, 2)$-connected, and satisfies (3.4). During this step, we try to make G_2 3-vertex-connected by switching some edges in F_2 while preserving the ℓ-edge-connectivity. In [8, Property 3.2], some sufficient conditions for two edges $e_1, e_2 \in F_2$ that are disconnected by some disconnecting pair and satisfies $\kappa(G_2 - \{e_1, e_2\}) \geq 2$, are given to admit a feasible switching such that at least one new pair of vertices in G_2 becomes 3-vertex-connected. However, if an input multigraph G is not 2-vertex-connected (a situation not assumed in [8]), there may be an edge $e \in F_2$ such that $G_2 - e$ is not 2-vertex-connected. Let $F^*(G_2) \subset F_2$ denotes the set of edges in F_2 such that the removal of any edge $e \in F^*(G_2)$ violates 2-vertex-connectivity of G_2.

Property 3. For each tight set T in G_2, $G_2[T \cup \Gamma_{G_2}(T)]$ contains at least one edge $e \in F_2 - F^*(G_2)$ with $T \cap V[e] \neq \emptyset$. □

Property 4. Let S be a disconnecting pair in G_2. If two edges $e_1, e_2 \in F_2 - F^*(G_2)$ satisfy $T_i \cap V[e_i] \neq \emptyset$, $i = 1, 2$, for two distinct S-components T_1 and T_2 in G_2, then $G_2 - \{e_1, e_2\}$ is 2-vertex-connected. □

From these properties, every tight set contains an edge in $F_2 - F^*(G_2)$ and the two edges $e_1, e_2 \in F_2 - F^*(G_2)$ satisfying one of the conditions in [8, Property 3.2] always satisfy $\kappa(G_2 - \{e_1, e_2\}) \geq 2$. Therefore we can repeat executing a feasible switching of pairs of edges in $F_2 - F^*(G_2)$ until none of the conditions in [8, Property 3.2] holds in G_2. Then it is not difficult to see that all disconnecting pairs in G_2 contain one common vertex, say v^*.

In [8, Property 3.3], another case in which a feasible switching can be performed is given. In this paper, we show a generalization of [8, Property 3.3] as it is now proved including the case of $\kappa(G_2 - \{e_1, e_2, e_3\}) \leq 1$.

If none of the conditions in [8, Properties 3.2, 3.3] for a feasible switching holds any longer, let $G_3 = (V, E \cup F_3)$ denote the resulting multigraph, where $F_3 = F_2$. If $\kappa(G_3) \geq 3$, we are done since $|F_3| = \lceil \alpha(G)/2 \rceil$ attains the lower bound $\gamma(G)$. Otherwise go to Step IV.

Step IV (Shifting edges):

Property 5. Let $S_i = \{v^*, v_i\}$, $i = 1, \cdots, q$, denote all disconnecting pairs in G_3. Then every $e \in F^*(G_3)$ satisfies $e = (v_i, v_j)$ for some $i \neq j$, or $e = (t, v_i)$ with $\Gamma_{G_3}(t) = \{v^*, v_i\}$ for some i. □

In [8, Property 3.4], some conditions are given to admit a feasible shifting of an edge $e_1 \in F_3 - F^*(G_3)$ incident to the common vertex v^* (and another edge $e_2 \in F_3 - F^*(G_3)$ such that $\kappa(G_3 - \{e_1, e_2\}) \geq 2$ holds and e_1 and e_2 are disconnected by some disconnecting pair in G_3, if necessary) that decreases the degree of v^* by one. From Property 5, $E_{G_3}(v^*) \cap F_3 \subseteq F_3 - F^*(G_3)$ follows. Furthermore, from Property 4, every two edges $e_1, e_2 \in F_3 - F^*(G_3)$ that are disconnected by some disconnecting pair satisfy $\kappa(G_3 - \{e_1, e_2\}) \geq 2$. Therefore we can apply [8, Property 3.4] to G_3, and repeat a feasible shifting and switching edges in $F_3 - F^*(G_3)$ until $|F_3 \cap E_{G_3}(v^*)| = 0$ or $c_{G_3}(v^*) = \ell$ holds.

Property 6. If $F^*(G_3) \neq \emptyset$ holds, we can execute a feasible switching or shifting of edges in $F_3 - F^*(G_3)$ by applying [8, Properties 3.2, 3,3, and 3.4]. □

Let $G_4 = (V, E \cup F_4)$ denote the resulting multigraph $G_3 = (V, E \cup F_3)$ obtained by repeating a feasible switching or shifting of edges in F_3 until none of conditions in [8, Properties 3.2, 3,3, and 3.4] holds in G_3. It holds $F^*(G_4) = \emptyset$ from Property 6. In this case, we observe that we can apply the latter part of Step IV and Step V of EV-AUGMENT3 [8] to G_4 (while maintaining condition $F^*(G_4) = \emptyset$). Then as observed in [8, Property 3.8], we see that an input multigraph G becomes $(\ell, 3)$-connected by adding $\min\{\lceil \alpha(G)/2 \rceil, \beta(G)\}$ new edges or by adding at most $2\ell - 3$ new edges. This establishes Theorem 2. □

4 Algorithm for Step I

We consider how to find a set F_1 such that $G_1 = (V \cup s, E \cup F_1)$ satisfies (3.1) – (3.3), and has a subpartition $\mathcal{X} = \{X_1, \cdots, X_{q_1}, X_{q_1+1}, \cdots, X_{q_2}\}$ of V, satisfying

$$\begin{array}{l} c_{G_1}(s, X_i) = \ell - c_G(X_i) \text{ for } i = 1, \cdots, q_1, \\ c_{G_1}(s, X_i) = 3 - |\Gamma_G(X_i)| \text{ and } V - X_i - \Gamma_G(X_i) \neq \emptyset \text{ for } i = q_1{+}1, \cdots, q_2, \\ \Gamma_{G_1}(s) \subseteq \cup_{X \in \mathcal{X}} X. \end{array} \quad (4.1)$$

Note that such an edge set F_1 attains $|F_1| = \alpha(G)$.

Algorithm ADD-EDGE*

1. After adding a sufficiently large number (say, ℓ) of edges between s and each vertex $v \in V$, discard them one by one as long as (3.1), (3.2) or (3.3) is not violated. Let $G_1' = (V \cup s, E \cup F_1')$ be the resulting multigraph where $F_1' = E_{G_1'}(s, V)$. Unless this G_1' has a subpartition of V satisfying (4.1), we continue shifting or removing edges in $E_{G_1'}(s, V)$, while preserving (3.1) – (3.3).

Property 7. For each edge $e = (s, t) \in E_{G_1'}(s, V)$ such that $G_1' - e$ violates (3.2) or (3.3), G_1' has a cut $T \subset V$ with $t \in T$ and $V - T - \Gamma_G(T) \neq \emptyset$, satisfying

(I) $|\Gamma_G(T)| = 2$ and $E_{G_1'}(s, T) = \{(s, t)\}$, or
(II) $|\Gamma_G(T)| = 1$ and $c_{G_1'}(s, t) = c_{G_1'}(s, u) = 1$ hold for $t \neq u \in T$ if $|T| \geq 2$, or $|\Gamma_G(t)| = 1$ and $c_{G_1'}(s, t) = 2$ if $T = \{t\}$. □

If $T \subseteq V$ satisfies (I) or (II) in Property 7, T is called *κ-critical.* Let $\mathcal{T}_1$ (resp., $\mathcal{T}_2$) denote the family of all κ-critical cuts T of type (I) (resp., (II)) such that no $T' \subset T$ is of type (I) (resp., (II)). For each edge $e = (s, u) \in F_1'$ such that $G_1' - e$ violates (3.1), there is a unique cut $X_u \subset V$, called *λ-critical*, such that $u \in X_u$, $c_{G_1'}(X_u) = \ell$ and $c_{G_1'}(Y) > \ell$ holds for all cuts Y with $u \in Y \subset X_u$. Let $\mathcal{X}_1$ denotes the family all λ-critical cuts X_u.

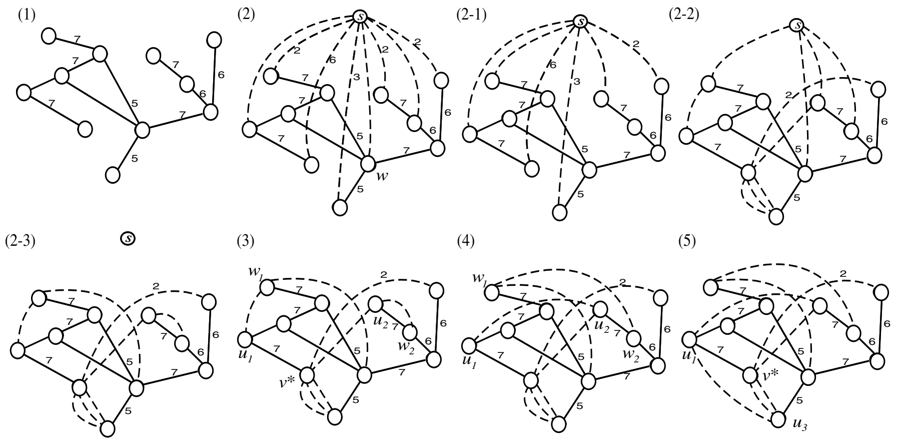


Fig. 1. Computational process of algorithm EV-AUGMENT3* for $\ell = 8$. (1) An input multigraph $G = (V, E)$ with $\lambda(G) = \kappa(G) = 1$, where the number beside each edge is the multiplicity of the edge (the numbers for multiplicity 1 are omitted). The two lower bounds in Section 2 are $\lceil \frac{\alpha(G)}{2} \rceil = \frac{18}{2} = 9$ and $\beta(G) = 3$. (2) $G_1 = (V \cup \{s\}, E \cup F_1)$ obtained by Step I. Edges in F_1 are drawn as broken lines. Now G_1 satisfies (3.1) for $\ell = 8$, (3.2) and (3.3), and the edge $\hat{e} = (s, w)$ is extra. (2-1) $G'_1 = (V \cup \{s\}, E \cup F'_1)$ with $F'_1 \subseteq F_1$ in Step II, satisfying (3.1) and (5.1). (2-2) $G'_2 = (V \cup \{s\}, E \cup (F_1 - F'_1) \cup F'_2)$ in Step II, where F'_2 is a set of edges obtained from splitting edges in F'_1. Now $G'_2[V]$ is $(8, 2)$-connected and every edge in $E_{G'_2}(s, V) - \hat{e}$ is κ-critical. (2-3) $G''_2 = (V \cup \{s\}, E \cup F_2)$ in Step II obtained from splitting edges in $F_1 - F'_1$ in G'_2, which creates no self-loop. (3) $G_2 = (V, E \cup F_2)$ obtained by Step II. The G_2 satisfies $\lambda(G_2) \geq 8$ but has a disconnecting pair $S = \{v^*, v\}$. (4) $G_2^{(1)} = (V, E \cup F_2^{(1)})$ obtained from G_2 by a feasible switching $\{(u_1, u_2), (w_1, w_2)\}/\{(u_1, w_1), (u_2, w_2)\}$ in Step III. Moreover, any switching is no longer feasible in $G_2^{(1)}$. (5) $G_4 = (V, E \cup F_4)$ obtained by shifting $\{(u_1, u_3)\}/\{(v^*, u_3)\}$ in Step IV. This G_4 is $(8, 3)$-connected. □

Property 8. Let $\mathcal{T}'_1 := \mathcal{T}_1 - \{T_i, T_j \in \mathcal{T}_1 \mid T_i$ and T_j cross each other in $G'_1\} - \{T_i \in \mathcal{T}_1 \mid T_i \subseteq T_j$ for some $T_j \in \mathcal{T}_2\}$. Then $\mathcal{X}_1 \cup \mathcal{T}'_1 \cup \mathcal{T}_2$ covers $\Gamma_{G'_1}(s)$, and every two cuts in $\mathcal{T}'_1 \cup \mathcal{T}_2$ are pairwise disjoint. □

From this property, if G'_1 does not have a subpartition of V satisfying (4.1), then there are two cuts $X \in \mathcal{X}_1$ and $T \in \mathcal{T}'_1 \cup \mathcal{T}_2$ satisfying the following:

$$\begin{aligned} &\text{(i)}\ \ c_{G'_1}(s, T) = 1 \text{ if } T \in \mathcal{T}'_1, \text{ and } c_{G'_1}(s, T) = 2 \text{ if } T \in \mathcal{T}_2. \\ &\text{(ii)}\ T \cap X \neq \emptyset \text{ and } (T - \textstyle\sum_{X_i \in \mathcal{X}_1} X_i) \cap \Gamma_{G'_1}(s) \neq \emptyset. \end{aligned} \tag{4.2}$$

2. Let H denote an arbitrary 2-vertex-connected multigraph. Let us regard $\mathcal{T}_1'$ as the family $\mathcal{T}(H)$ of all minimal tight sets in H, since every two cuts in $\mathcal{T}_1'$ are pairwise disjoint and every cut $T \in \mathcal{T}_1'$ satisfies $|\Gamma_G(T)| = 2$ and $V - T - \Gamma_{G_1'}(T) \neq \emptyset$. It was shown in the algorithm ADD-EDGE in [8] that the number of two cuts $X \in \mathcal{X}_1$ and $T \in \mathcal{T}(H)$ satisfying (4.2) in H can be decreased to 0. Hence, by applying the same algorithm, the number of pairs of cuts $X \in \mathcal{X}_1$ and $T \in \mathcal{T}_1'$ satisfying (4.2) can be decreased to 0. Moreover, we can see that the resulting multigraph G_1'' has a subpartition of V satisfying (4.1) and hence let $G_1 := G_1''$. □

5 Algorithm for Step II

Algorithm SPLIT

1. Let $F_1' \subseteq F_1$ be a set of edges such that $G_1' = (V \cup s, E \cup F_1')$ satisfies (3.1) and the following (5.1), but no $(V \cup s, E \cup F')$ with $F' \subset F_1'$ satisfies them.

$$\begin{aligned}|\Gamma_G(X)| + |\Gamma_{G_1'}(s) \cap X| \geq 2 \ \text{ for all cuts } X \subset V \text{ with } V - X - \Gamma_G(X) \neq \emptyset \\ \text{and } |\Gamma_G(X)| + |X| \geq 2, \\ c_{G_1'}(s, x) \geq 2 \ \text{ for all } X = \{x\} \text{ with } \Gamma_G(X) = \emptyset.\end{aligned} \tag{5.1}$$

If $c_{G_1'}(s)$ is odd, then we choose one edge $e^* \in F_1 - F_1'$, and update $G_1' := G_1' + e^*$ and $F_1' := F_1' + e^*$. Then it is shown in [7] that by a complete splitting at s in G_1', which does not create a self-loop, we can obtain a multigraph $G_2' = (V \cup \{s\}, E \cup (F_1 - F_1') \cup F_2')$ satisfying one of the following conditions, where F_2' is the set of edges obtained from splitting edges in F_1'.

(i) $G_2'[V]$ is $(\ell, 2)$-connected.
(ii) $G_2'[V]$ is $(\ell$,1)-connected and has exactly one disconnecting vertex v. At most one edge in F_2' is incident to v, and each edge $e \in F_2'$ which is not incident to v satisfies $p(G_2'[V] - v) = p((G_2'[V] - e) - v) - 1$.

If G_2' satisfies (ii), go to step 2; otherwise go to step 3.

2. Let T_i, $i = 1, \cdots, r'$, denote all v-components in $G_2'[V]$. If $G_2'[V]$ has the edge $e = (v, t_1) \in F_2'$ incident to v, then assume $t_1 \in T_1$ without loss of generality. Here we can prove from the above property of edges in F_2' and properties (3.2) and (3.3) in G_1 that $c_{G_2'}(s, T_1) \geq 1$ and $c_{G_2'}(s, T_i) \geq 2$ hold for $i = 2, \cdots, r'$. Let $G_2'' = (V \cup \{s\}, E \cup (F - F_1' - F_1'') \cup F_2' \cup F_2'')$ be the multigraph resulting from a sequence of splitting pairs of edges (s, a_i) and (s, b_{i+1}) to (a_i, b_{i+1}), $i = 1, \cdots, r' - 1$, where $a_1 \in T_1 \cap \Gamma_{G_2'}(s)$, $a_i, b_i \in T_i \cap \Gamma_{G_2'}(s)$ for $i = 2, \cdots, r'$, $F_1'' = \{\{(s, a_i), (s, b_{i+1})\} \mid i = 1, \cdots, r' - 1\}$, and $F_2'' = \{(a_i, b_{i+1}) \mid i = 1, \cdots, r' - 1\}$. This $G_2''[V]$ is now $(\ell, 2)$-connected. Let $G_2' := G_2''$, $F_1' := F_1' \cup F_1''$, and $F_2' := F_2' \cup F_2''$, and go to 3.

3. Now $G_2'[V]$ is $(\ell, 2)$-connected and $c_{G_2'}(s)$ is even. Here it is not difficult to see that G_2' has a complete splitting at s which creates no self-loop (if necessary, the extra edge $\hat{e}$ chosen in Step I will be rechosen). Let $G_2 := G_2'' - s$, where G_2'' denotes the multigraph resulting from such a complete splitting in G_2'. □

6 Concluding Remarks

In this paper, we combined the algorithms EV-AUGMENT [7] and EV-AUGMENT3 [8], and gave an algorithm for augmenting a given arbitrary graph G to an ℓ-edge-connected and 3-vertex-connected graph by adding the smallest number of new edges. However, our lower bound on the optimal value does not always give the exact optimal value. So, it is desired to find a new and stronger lower bound on the optimal value.

References

1. K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM J. Compt., **5** 1976, 653–665. 160
2. A. Frank, *Augmenting graphs to meet edge-connectivity requirements*, SIAM J. Disc. Math., **5** 1992, 25–53. 160
3. T. Hsu and V. Ramachandran, *A linear time algorithm for triconnectivity augmentation*, Proc. 32nd IEEE Symp. Found. Comp. Sci., 1991, 548–559. 160
4. T. Hsu and V. Ramachandran, *Finding a smallest augmentation to biconnect a graph*, SIAM J. Compt., **22** 1993, 889–912. 160
5. T. Hsu, *Undirected vertex-connectivity structure and smallest four-vertex-connectivity augmentation*, LNCS 920, Springer-Verlag, 6th ISAAC, 1995, pp.274–283. 160
6. T. Hsu and M. Kao, *Optimal bi-level augmentation for selectively enhancing graph connectivity with applications*, LNCS 1090, Springer-Verlag, 2nd COCOON, 1996, 169–178. 159, 160
7. T. Ishii, H. Nagamochi, and T. Ibaraki, *Augmenting edge-connectivity and vertex-connectivity simultaneously*, LNCS, Springer-Verlag, 8th ISAAC 1997, pp. 102-111. 159, 160, 168, 169
8. T. Ishii, H. Nagamochi, and T. Ibaraki, *Optimal augmentation to make a graph k-edge-connected and triconnected*, Proc. of 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 280-289. 159, 160, 165, 166, 168, 169
9. L. Lovász, Combinatorial Problems and Exercises, North-Holland, 1979. 162
10. T. Watanabe and A. Nakamura, *Edge-connectivity augmentation problems*, J. Comp. System Sci., **35** 1987, 96–144. 160
11. T. Watanabe and A. Nakamura, *A smallest augmentation to 3-connect a graph*, Disc. Appl. Math., **28** 1990, 183–186. 160

Polyhedral Structure of Submodular and Posi-modular Systems[⋆]

Hiroshi Nagamochi and Toshihide Ibaraki

Kyoto University, Kyoto, Japan 606-8501
{naga,ibaraki}@kuamp.kyoto-u.ac.jp

Abstract. Let V be a finite set, and $\Re$ be the set of reals. We consider the polyhedron $P = \{z \in \Re^V_- \mid \sum_{i \in X} z(i) \le f(X),\ \forall X \in 2^V\}$ for a system (V, f) with an intersecting submodular and posi-modular set function $f : 2^V \mapsto \Re$, where $\Re^V_-$ denotes the set of $|V|$-dimensional nonpositive vectors. We first prove that there is a laminar family $\mathcal{X} \subseteq 2^V$ such that P is characterized by $\{z \in \Re^V_- \mid \sum_{i \in X} z(i) \le f(X),\ \forall X \in \mathcal{X}\}$. Based on this, we can solve in polynomial time the edge-connectivity augmentation problem with an additional constraint that the number of vertices to which new edges are incident is minimized.

1 Introduction

Let V be a finite set, where $|V|$ is denoted by n. A singleton set $\{v\}$ may be written as v. For two subsets $X, Y \subseteq V$, we say that X and Y *intersect* each other if $X \cap Y \ne \emptyset$, $X - Y \ne \emptyset$ and $Y - X \ne \emptyset$. A family $\mathcal{X} \subset 2^V$ is called *laminar* if no two subsets in $\mathcal{X}$ intersect each other. Two intersecting sets X and Y are called *crossing* if $V - (X \cup Y) \ne \emptyset$ also holds.

Let $\Re$ (resp., $\Re_+$ and $\Re_-$) be the set of reals (resp., nonnegative reals and nonpositive reals), and let $\Re^V$ (resp., $\Re^V_+$ and $\Re^V_-$) be the set of n-dimensional real vectors (resp., nonnegative real vectors and nonpositive real vectors) on a ground set V. A *set function* f on V is a function $f : 2^V \mapsto \Re$. For a vector $z \in \Re^V$ and a subset $X \subseteq V$, we denote $\sum_{i \in X} z(i)$ by $z(X)$. Such a function $z : 2^V \mapsto \Re$ is called *modular*. A function f is called *fully (resp., intersecting, crossing) submodular* if it satisfies the following inequality

$$f(X) + f(Y) \ge f(X \cap Y) + f(X \cup Y) \tag{1}$$

holds for every (resp., intersecting, crossing) pair of sets $X, Y \subseteq V$. A function f is called *fully (resp., intersecting, crossing) supermodular* if $-f$ is fully (resp., intersecting, crossing) submodular. An f is called *symmetric* if $f(X) = f(V - X)$ holds for all $X \subseteq V$. In this paper, we call a function f *fully (resp., intersecting, crossing) posi-modular* if

$$f(X) + f(Y) \ge f(X - Y) + f(Y - X) \tag{2}$$

⋆ This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan, and the subsidy from the Inamori Foundation.

holds for every (resp., intersecting, crossing) pair of sets $X, Y \subseteq V$ [6]. An f is called *fully (resp., intersecting, crossing) nega-modular* if $-f$ is fully (resp., intersecting, crossing) posi-modular. Any modular function z such that $z(i) \geq 0$ for all $i \in V$ is posi-modular. Also a symmetric fully submodular function f is fully posi-modular. However, the converse is not generally true.

A pair (V, f) of a finite set V and a set function f on V is called a *system*. The optimization in a system (V, f) has been much studied, such as:

$$\textbf{Problem 1 (primal type)}: \begin{array}{lll} \text{minimize} & \Phi(z) & \\ \text{subject to} & z(X) \leq f(X) & \text{for all } X \in 2^V \\ & 0 \leq z(i) \leq d(i) & \text{for all } i \in V \end{array}$$

(an additional constraint $z(V) = f(V)$ may also be imposed), where $\Phi(z) : R^V \mapsto \Re$ is an objective function and $d \in \Re_+^V$ is a given constant vector. For fully submodular functions f, Problem 1 appears in many applications [5]. Given a system (V, g), a dual type of this problem is stated as follows:

$$\textbf{Problem 2 (dual type)}: \begin{array}{lll} \text{minimize} & \Phi(t) & \\ \text{subject to} & g(X) \leq t(X) & \text{for all } X \in 2^V \\ & 0 \leq t(i) \leq d(i) & \text{for all } i \in V \end{array}$$

(where we may also impose an additional constraint $t(V) = g(V)$). Problem 2 with a certain supermodular function g appears in the edge-connectivity augmentation problem [1,3] and the problem of computing the core of a convex game [8]. The above Problems 1 and 2 are generalized into the following common formulation. For two set functions g and f on V, and vectors $d_1, d_2 \in \Re_+^V$, we consider:

$$\textbf{Problem 3 (mixed type)}: \begin{array}{lll} \text{minimize} & \Phi(z) & \\ \text{subject to} & g(X) \leq z(X) \leq f(X) & \text{for all } X \in 2^V \\ & d_1(i) \leq z(i) \leq d_2(i) & \text{for all } i \in V. \end{array}$$

In this paper, we consider Problems 1-3 with intersecting submodular and posi-modular functions f and $-g$.

Before going into details, let us explain an application of Problem 2 to the edge-connectivity augmentation problem. Let $N = (V, E, c)$ be an undirected complete network with a vertex set V, an edge set $E = V \times V$ and an edge weight function $c : E \mapsto \Re_+$. The *cut function* $f_N : 2^V \mapsto \Re_+$ is defined by $f_N(X) = \sum\{c(e) \mid e = \{u, v\} \in E,\ u \in X,\ v \in V - X\}$ (where $f_N(\emptyset) = f_N(V) = 0$). It is known (and easy to see) that the cut function f_N is symmetric and fully submodular. The *edge-connectivity augmentation problem* asks to increase edge weights c to obtain a k-edge-connected network N' (i.e., $f_{N'}(X) \geq k$ holds for all $X \in 2^V - \{\emptyset, V\}$). Frank [3] introduced an additional constraint to this problem, *the degree constraint*: Given a vector $d \in \Re_+^V$, the output k-edge-connected network N' is required to satisfy $\sum_{e \in E(i)}(c'(e) - c(e)) \leq d(i)$ for all $i \in V$, where $E(i)$ denotes the set of edges incident to a vertex i. The problem can be formulated as Problem 2 by using the following result.

Lemma 1. [1,3] *Given a network $N = (V, E, c)$, a constant $k \geq 0$, and a vector $t \in \Re_+^V$ such that*

$$f_N(X) + t(X) \geq k \text{ for all } X \in 2^V - \{\emptyset, V\}, \tag{3}$$

there is a k-edge-connected network $N' = (V, E, c')$ satisfying $\sum_{e\in E(i)}(c'(e) - c(e)) = t(i)$ for all $i \in V$. Also, the c' can be chosen as integers if c, t and $k \geq 2$ are integers and $t(V) = \sum_{i\in V} t(i)$ is an even integer. □

Notice that the total increase $\sum_{e\in E}(c'(e) - c(e))$ of weights is $\frac{1}{2}t(V)$. Therefore, in order to solve the edge-connectivity augmentation problem, we only need to find a vector $t \in \Re_+^V$ that minimizes $t(V) = \sum_{i\in V} t(i)$ among all vectors t satisfying (3) (and $t(i) \leq d(i)$, $i \in V$ if the degree constraint is imposed). Hence, by defining $\Phi(t) = \frac{1}{2}t(V)$ and a fully supermodular set function g by $g(X) = k - f_N(X)$ for all $X \in 2^V$, we see that the smallest amount $\alpha(N, k)$ of new weights to be added to obtain a k-edge-connected network N' is given by the minimum value of $\Phi(t)$ over all $t \in \Re_+^V$ satisfying $g(X) \leq t(X)$ for all $X \in 2^V - \{\emptyset, V\}$ (and $t(i) \leq d(i)$, $i \in V$ if the degree constraint is imposed). In the case of integer version, $\alpha(N, k)$ is given by $\lceil \frac{1}{2}\Phi(t) \rceil$. In any case, the problem of finding such a vector t can be formulated as Problem 2 with these Φ, g and d.

In this paper, we first characterize the polyhedra of Problems 1-3 with intersecting submodular and posi-modular functions f and $-g$, and then present a combinatorial algorithm for solving Problem 3 with a linear function $\Phi(t)$ (and assuming a further restriction on g). Note that Problem 3 is more general than Problem 2 in the sense that it allows additional constraints $z(X) \leq f(X)$, $X \in 2^V$. This enables us to solve in polynomial time the edge-connectivity augmentation problem with a more general degree constraint that, for each subset $X \subset V$, the total increase of degrees in X in the resulting network N' is bounded by a given constant $f(X)$.

We also show that Problem 2 can be solved for an objective function $\Phi(t) = |\{i \in V \mid t(i) > 0\}|$ in $O(n^3)$ function value oracle calls. Based on this, we can solve in polynomial time the problem of augmenting edge-connectivity of a network so as to minimize the number of vertices having edges whose weights are increased.

The paper is organized as follows. In Section 2, we characterize the polyhedron of Problem 2, and give algorithms for solving Problems 1-3. In Section 3, we characterize all the extreme points of the base polyhedron of Problem 2, and discuss a relation to the core of a convex game.

2 Polyhedral Structures and Problems 1 - 3

A *polyhedron* of a system (V, f) is defined by

$$P(f) = \{z \in \Re^V \mid z(X) \leq f(X),\ \forall X \in 2^V - \{\emptyset, V\}\}, \tag{1}$$

where $X = \emptyset$ and V are not considered in the definition, and a *base polyhedron* of (V, f) by

$$B(f) = \{z \in P(f) \mid z(V) = f(V)\}, \tag{2}$$

where possibly $B(f) = \emptyset$. Let $P_-(f)$ and $B_-(f)$ denote $P(f) \cap \Re^V_-$ and $B(f) \cap \Re^V_-$, respectively, and let $P_+(f)$ and $B_+(f)$ denote $P(f) \cap \Re^V_+$ and $B(f) \cap \Re^V_+$, respectively. For two set functions f_1 and f_2 on V, we denote by $(f_1 - f_2)$ the set function f' with $f'(X) = f_1(X) - f_2(X)$ for all $X \in 2^V$.

We say that a subset $X \subset V$ *separates* x and y if $|\{x, y\} \cap X| = 1$. For a submodular system (V, f), an ordered pair (x, y) of elements in V is called a *pendant pair* if $f(x) \le f(X)$ holds for all sets $X \subset V$ that separate x and y.

Lemma 2. [6] *For an intersecting submodular and posi-modular set function f on V (where $n = |V| \ge 2$), there exists a pendant pair (x, y). Furthermore, such a pendant pair can be obtained by using $O(n^2)$ function value oracle calls.* □

2.1 Polyhedral Structure of $P_-(f)$

In this section, we first consider the set of all feasible vectors to Problem 2, where we assume that g is an intersecting supermodular and nega-modular set function and a vector $d \in \Re^V_+$ is given by $d(i) = +\infty$ $(i \in V)$. In this case, $f = -g$ is intersecting submodular and posi-modular. Then a vector t is feasible to Problem 2 if and only if $-t \in P_-(f)$ holds for a system (V, f).

We now prove that, given a system (V, f) with an intersecting submodular and posi-modular set function f, there is a laminar family $\mathcal{X} \subseteq 2^V - \{\emptyset, V\}$ that characterizes $P_-(f)$ as follows.

$$P_-(f) = P_-(f; \mathcal{X}), \tag{3}$$

where we use notations $P(f; \mathcal{X}) = \{z \in \Re^V \mid z(X) \le f(X) \text{ for all } X \in \mathcal{X}\}$ and $P_-(f; \mathcal{X}) = P(f; \mathcal{X}) \cap \Re^V_-$.

Given an intersecting submodular and posi-modular set function f on V, we compute the above laminar family $\mathcal{X}$ as follows. Initially we set $\mathcal{X} := \emptyset$ and $z(i) := 0$ for all $i \in V$. Then, for each $i \in V$, we check whether $z(i) \le f(i)$ (i.e., $f(i) \ge 0$) holds or not. If $f(i) < 0$ then we reset $z(i)$ by $z(i) := f(i)$ and add $\{i\}$ to $\mathcal{X}$. Now $z(i) \le f(i)$ (i.e., $(f - z)(i) \ge 0$) holds for all $i \in V$. Note that $f - z$ remains to be intersecting submodular and posi-modular. Hence there is a pendant pair (x, y) in system $(V, f - z)$ by Lemma 2, for which any cut X separating x and y satisfies $z(X) \le f(X)$. Then we can contract x and y into a single element x^* without losing any cut X that satisfies $z(X) > f(X)$. After this contraction, we check whether the new element x^* satisfies $(f - z)(x^*) \ge 0$. If $(f - z)(x^*) < 0$, then we add to $\mathcal{X}$ the set X^* of all elements which have been contracted into x^* so far, and decrease $z(i)$ of some $i \in X^*$ so that $z(X^*) = f(X^*)$ holds (where more than one $z(i)$ may be decreased as long as $z(X^*) = f(X^*)$ is satisfied). (If $(f - z)(x^*) \ge 0$, no $z(i)$ is changed.) Then we repeat finding a new pendant pair and contracting them into a single element in the resulting system, until the system has only one element. The entire algorithm is described as follows.

Algorithm LAMINAR

Input: A system (V, f) with an intersecting submodular and posi-modular set function f on V,
where $n = |V| \geq 2$.
Output: A vector $z \in P_-(f)$, and a laminar $\mathcal{X}$ of V satisfying (3).

```
1  $\mathcal{X} := \emptyset$; $z(i) := 0$ for all $i \in V$;
2  For each $i \in V$, if $f(i) < 0$ then $z(i) := f(i)$ and $\mathcal{X} := \mathcal{X} \cup \{\{i\}\}$;
3  **for** $i := 1$ to $n-1$ **do**
4     Find a pendant pair $(x, y)$ in $(V', f-z)$;
5     Let $(V', f-z)$ again denote the system obtained from the current
      $(V', f-z)$ by contracting $x$ and $y$ into a single element $x^*$;
6     **if** $(f-z)(x^*) < 0$ **then**
7        Let $X^*$ be the set of all elements of $V$ which have been contracted
         to $x^*$;
8        Decrease $z(i)$, $i \in X^*$ arbitrarily so that the resulting $f - z$ satisfies
         $(f-z)(X^*) = 0$ in $(V, f)$;
9        $\mathcal{X} := \mathcal{X} \cup \{X^*\}$
10    **end** /* if */
11 **end**. /* for */
```

Clearly, LAMINAR runs in $O(n^3)$ function value oracle calls, as the pendant pair in line 4 can be found in $O(n^2)$ function value oracle calls by Lemma 2. Note that the vector z output by LAMINAR may not be unique because there are many ways of decreasing $z(i)$, $i \in X^*$ in line 8. Let $OUTPUT(f)$ denote the set of all vectors z that can be output by LAMINAR for a given input f.

For a laminar family $\mathcal{X} \subseteq 2^V$ on V, a subset $Y \in \mathcal{X}$ is called a *child* of a subset $X \in \mathcal{X}$ (and the X is called the *parent* of Y) if $Y \subset X$ and there is no other subset $Y' \in \mathcal{X}$ with $Y \subset Y' \subset X$. For a subset $X \in \mathcal{X}$, let $ch(X)$ denote the set of children of X, and $pa(X)$ denote the parent of X (possibly $pa(X) = \emptyset$).

Let $\mathcal{X}$ be a family of subsets of V output by LAMINAR, which is clearly laminar. We represent $\mathcal{X}$ by a rooted tree as follows. Define the laminar family $\mathcal{V} = \mathcal{X} \cup \{V\} \cup \{\{i\} \mid i \in V\}$ and define the tree $T = (\mathcal{V}, \mathcal{E})$ on $\mathcal{V}$, where the parent-child relation in the tree is given by $pa(X)$ and $ch(X)$. Clearly V is the root of T. Define $f' : \mathcal{V} \mapsto \Re$ by $f'(X) = 0$ if $X = \{i\}$ and $f(i) \geq 0$; $f'(X) = f(X)$ otherwise. From the behavior of LAMINAR, the next properties are observed.

Lemma 3. *For a system (V, f) with an intersecting submodular and posi-modular set function f on V with $|V| \geq 2$, let z and $\mathcal{X}$ be the vector and the laminar family output by algorithm* LAMINAR. *Let the tree T be defined as above. Then:*

(i) *$z \in P_-(f)$ (hence $OUTPUT(f) \subseteq P_-(f)$).*
(ii) *For each non-root vertex X in T, $f'(X) < \sum_{Y \in ch(X)} f'(Y)$ holds.* □

From this lemma, we can prove (3). Clearly, $P_-(f; \mathcal{X}) \supseteq P_-(f)$. The converse is shown as follows. Given a vector $z' \in P_-(f)$, there is a way of decreasing

$\{z(i) \mid i \in X^*\}$ in line 8 of LAMINAR so that the algorithm outputs a vector $z(\in OUTPUT(f))$ such that $z \geq z'$. Hence $z' \in P_-(f)$ follows (further detail is omitted).

Theorem 1. *For a system (V, f) with an intersecting submodular and posi-modular set function f on V with $|V| \geq 2$, let $\mathcal{X}$ be the laminar family output by algorithm* LAMINAR*. Then:*

(i) $P_-(f) = P_-(f; \mathcal{X})$.
(ii) $B_-(f) \neq \emptyset$ *if and only if* $f'(V) \leq \sum_{Y \in ch(V)} f'(Y)$ *holds.* □

2.2 Problem 2

Based on Theorem 1, we can solve Problem 2, if it has a linear objective function $\Phi(t) = \sum_{i \in V} w(i)t(i)$ defined by a cost vector $w \in \Re^V$. The proof for this case will be given in Section 2.4 under a more general setting of Problem 3. We consider another special case of Problem 2 when the objective function is given by

$$\Phi(t) = |\{i \in V \mid t(i) > 0\}| \text{ (i.e., the number of nonzero entries).}$$

By Theorem 1, we prove the next property (the proof is omitted).

Theorem 2. *For an intersecting supermodular and nega-modular set function g on V, and a vector $d \in \Re^V_+$, Problem 2 with an objective function $\Phi(t) = |\{i \in V \mid t(i) > 0\}|$ can be solved by using $O(n^3)$ function value oracle calls. If Problem 2 is feasible, then there is a feasible vector t which minimizes $\Phi(t)$ and $\Phi'(t) = \sum_{i \in V} t(i)$ at the same time. Such a solution t also can be found by using $O(n^3)$ function value oracle calls.* □

By applying this and Lemma 1 to the degree constrained edge-connectivity augmentation problem of a graph to minimize the number of vertices whose incident edges have increased weights, we obtain the next (the proof is omitted).

Theorem 3. *For a complete network $N = (V, E = V \times V, c)$, $k \geq 2$ and $d \in \Re^V_+$, where all $c(e)$, k and $d(i)$ are integers, let $c'(\geq c)$ be a new integer-valued edge weight function such that $N' = (V, E, c')$ is k-edge-connected under the degree constraint that new degree of each vertex $i \in V$ is at most $d(i)$. There is a c' that simultaneously minimizes* (i) *the number of vertices to which an edge with increased weight is incident and* (ii) *the total amount of increase $\sum_{e \in E}(c'(e) - c(e))$. Such a c' can be found in $O((nm + n^2 \log n) \log n)$ time, where $n = |V|$ and m is the number of edges of positive weights in N.* □

2.3 Problem 1

In this subsection, we consider polyhedra $P(f)$ and $P_+(f)$ for an intersecting submodular and posi-modular function f on V, which appear in Problem 1. However, we do not consider the constraint $z \leq d$, as this more general case will be considered in the next subsection as Problem 3.

To generalize Theorem 1 to this case, we further assume that the set function $\hat{f}$ defined by

$$\hat{f}(X) = f(X) - m_f(X) \text{ for } X \in 2^V$$

is intersecting submodular and posi-modular, where m_f denotes the modular function on V defined from f by $m_f(i) = f(i)$ for all $i \in V$.

Now we discuss how to compute a vector $z \in P_+(f)$. Let us consider $y = z - m_f$. Then $0 \le z(X) \le f(X)$ holds if and only if $0 \le y(X) + m_f(X) \le f(X) = \hat{f}(X) - m_f(X)$. Thus the problem is equivalent to finding a vector $y \in P(\hat{f}) \cap \{y \in \Re^V \mid y + m_f \ge 0\}$.

We first consider $P(f)$. Note that $P(\hat{f}) = P_-(\hat{f})$ since $\hat{f}(i) = 0$ holds for all $i \in V$. Therefore, $P(f) = \{z = y + m_f \mid y \in P_-(\hat{f})\}$. By applying Theorem 1 to system $(V, \hat{f})$, we obtain a laminar family $\mathcal{X}$ such that $P_-(\hat{f}) = P_-(\hat{f}; \mathcal{X})$, by using $O(n^3)$ function value oracle calls. Clearly, for any $z \in P(f; \mathcal{X})$, we have $y = z - m_f \in P_-(\hat{f}; \mathcal{X}) = P_-(\hat{f})$, and hence $z \in P(f)$ (the converse is also clear). Thus, $P(f) = P(f; \mathcal{X})$ holds.

Next consider a vector $z \in P_+(f)$. From the above argument, it holds $P_+(f) = P(f) \cap \Re^V_+ = \{z = y + m_f \mid y \in P_-(\hat{f}),\ y \ge -m_f\}$. A vector $y \in P_-(\hat{f})$ satisfying constraint $y \ge -m_f$ (if any) can be easily computed. From these, we establish the next result.

Theorem 4. *For a system (V, f) with a set function f on V with $n = |V| \ge 2$ such that $f - m_f$ is intersecting submodular and posi-modular, there is a laminar family $\mathcal{X}$ such that $P(f) = P(f; \mathcal{X})$. Such a family $\mathcal{X}$ and a vector $z \in P_+(f)$ (if any) can be found by $O(n^3)$ function value oracle calls.* □

2.4 Problem 3

In this subsection, we solve Problem 3 with a linear objective function $\Phi(z) = \sum_{i \in V} w(i)z(i)$ for a given vector $w \in \Re^V$.

Theorem 5. *Let g and f be set functions on V, and $d_1, d_2 \in \Re^V_+$ and $w \in \Re^V$. If $-g$ and $f - m_f$ are both intersecting submodular and posi-modular, then an optimal solution z to Problem 3 with objective function $\Phi(z) = \sum_{i \in V} w(i)z(i)$ can be found by using $O(n^3)$ function value oracle calls and by solving a minimum cost flow problem with $O(n)$ vertices and arcs.*

Proof Sketch: By Theorems 1 and 4, there are laminar families $\mathcal{X}_1$ and $\mathcal{X}_2$ such that $\{z \in \Re^V_+ \mid g(X) \le z(X) \text{ for all } X \in \mathcal{X}_1\} = \{z \in \Re^V_+ \mid g(X) \le z(X) \text{ for all } X \in 2^V - \{\emptyset, V\}\}$ and $\{z \in \Re^V \mid z(X) \le f(X) \text{ for all } X \in \mathcal{X}_2\} = \{z \in \Re^V \mid z(X) \le f(X) \text{ for all } X \in 2^V - \{\emptyset, V\}\}$. Thus, the problem is restated as

$$\begin{array}{lll} \text{minimize} & \Phi(z) = \sum_{i \in V} w(i)z(i) & \\ \text{subject to} & g(X) \le z(X) & \text{for all } X \in \mathcal{X}'_1 \\ & z(X) \le f(X) & \text{for all } X \in \mathcal{X}'_2 \\ & d_1(i) \le z(i) \le d_2(i) & \text{for all } i \in V, \end{array}$$

where $\mathcal{X}'_i = \mathcal{X}_i \cup \{\emptyset, V\}$ for $i = 1, 2$. Then it is not difficult to see that the problem can be formulated as the minimum cost flow problem in a directed network (further detail is omitted). $\square$

3 Extreme Points of Base Polyhedron

In this section, we characterize all extreme points of a base polyhedron $B_-(f)$ defined for an intersecting submodular and posi-modular set function f. We then show some relation of the result to a core in a convex game.

3.1 All Extreme Points of $B_-(f)$

Let Π_n be the set of all permutations of $(1, 2, \ldots, n)$. For a subset $P \subseteq \Re^V_-$ and a permutation $\pi \in \Pi_{|V|}$, a vector $z \in \Re^V_-$ is called *lexicographically π-minimal* (*π-minimal*, for short) in P if there is no other vector $z' \in P$ which is lexicographically smaller than z with respect to π; i.e., there is no j such that $z'(\pi(i)) = z(\pi(i))$ for $i = 1, 2, \ldots, j-1$ and $z'(\pi(j)) < z(\pi(j))$. Let $L(f)$ be the set of π-minimal vectors in $B_-(f)$ for all $\pi \in \Pi_n$, and $EP(f)$ be the set of all extreme points in $B_-(f)$. Based on Theorem 1, we can show the next result.

Theorem 6. *Let (V, f) be a system with an intersecting submodular and posi-modular set function f on V with $n = |V| \geq 2$. If $B_-(f) \neq \emptyset$, then $L(f) = EP(f)$ holds.* $\square$

If $B_-(f) \neq \emptyset$ and $L(f) = EP(f)$, then we define the *mean vector* ψ_f of all π-minimal vectors z_π, $\pi \in \Pi_n$ by $\psi_f = \frac{1}{n!}\sum\{z_\pi \mid \pi \in \Pi_n\}$, where possibly $z_\pi = z_{\pi'}$ holds for two permutations $\pi, \pi' \in \Pi_n$. Again by Theorem 1, we can show that, for an intersecting submodular and posi-modular set function f, the mean vector ψ_f can be efficiently computed from the laminar family $\mathcal{X}$.

Theorem 7. *For a system (V, f) with an intersecting submodular and posi-modular set function f on V with $n = |V| \geq 2$, let $\mathcal{X}$ be the laminar family output by algorithm* LAMINAR. *Then the mean vector ψ_f of all lexicographically minimal vectors in $B_-(f)$ can be computed from $\mathcal{X}$ in $O(n^2)$ time.* $\square$

3.2 A Relation to a Convex Game

A cooperative game in the game theory is defined by a pair (V, g) of a set V of players and a nonnegative set function g on V, where g is called the characteristic function of the game and satisfies $g(\emptyset) = 0$. Several solution concepts such as core, Shapley value, τ-value and others have been proposed. The core of a game (V, g) is the set $CORE(g)$ of nonnegative vectors $z \in \Re^V_+$ such that $z(X) \geq g(X)$ for all $X \in 2^V$ and $z(V) = g(V)$. In other words, it can be defined by $CORE(g) = \{-z' \mid z' \in B_-(-g)\}$ for a system $(V, -g)$. Note that $CORE(g)$ is always a convex set. The problem of testing whether a convex game (V, g)

has a nonempty $CORE(g)$ can be solved by computing the minimum value $\Phi(z) = \sum_{i \in V} z(i)$ in Problem 2 with $d = +\infty$.

The Shapley value $\phi_g \in \Re^V$ is a solution concept proposed by Shapley [7], which is defined by

$$\phi_g(i) = \sum_{S \subseteq V:\ i \in S} \frac{(|S|-1)!(n-|S|)!}{n!}[g(S) - g(S-i)] \text{ for each } i \in V.$$

A game (V, g) is called *convex* if g is a *fully supermodular* set function on V. Several structural properties have been studied for a convex game. In this section, we consider a game (V, h) with an intersecting supermodular and nega-modular function h on V, which is slightly different from a convex game (V, g). We show that a game (V, h) has a considerably different structure from that of a convex game. Let us review some structural properties of a convex game.

Theorem 8. [2,8] *For a convex game (V, g), $CORE(g)$ is always nonempty. For any permutation $\pi \in \Pi_n$, the π-minimal vector $z_\pi \in \Re^V_+$ belongs to $CORE(g)$ and is given by $z_\pi(\pi(i)) = g(\{\pi(1), \pi(2), \ldots, \pi(i)\}) - g(\{\pi(1), \pi(2), \ldots, \pi(i-1)\})$ for $i = 1, 2, \ldots, n$. Moreover, the set of all extreme points of $CORE(g)$ is given by the set of π-minimal vectors z_π, $\pi \in \Pi_n$.* □

We denote by ψ_g the mean vector of all π-minimal vectors z_π, $\pi \in \Pi_n$, for a convex game (V, g).

Theorem 9. [7] *For a convex game (V, g), the Shapley value $\phi_g \in \Re^V$ is given by $\phi_g = \psi_g$.* □

As to computing the Shapley value, we easily observe the following intractability.

Lemma 4. *For a convex game (V, g), there is no algorithm that computes the Shapley value $\phi_g \in \Re^V$ by using less than $2^n - 1$ function value oracle calls, where $n = |V|$.* □

Now let us consider the counter part of the above results in a game (V, h) with an intersecting supermodular and nega-modular function h. By applying Theorems 1 and 6 to system $(V, -h)$, we have the next result.

Theorem 10. *For a game (V, h) with an intersecting supermodular and nega-modular function $h : 2^V \mapsto \Re_+$, $CORE(h)$ is nonempty if and only if $h(V) \geq \sum_{Y \in ch(V)} h(Y)$ holds, where $ch(V)$ is the set of maximal subsets X in the laminar family $\mathcal{X}$ obtained from $(V, -h)$ by algorithm* LAMINAR. *Moreover, the set of all extreme points of $CORE(h)$ is given by the set of π-minimal vectors z_π, $\pi \in \Pi_n$.* □

From Theorem 7, we obtain the following.

Theorem 11. *For a game (V, h) with an intersecting supermodular and nega-modular function $h : 2^V \mapsto \Re_+$, assume that $CORE(h) \neq \emptyset$. Then the mean vector ψ_h of all π-minimal vectors z_π, $\pi \in \Pi_n$, can be computed by using $O(n^3)$ function value oracle calls, where $n = |V|$.* $\square$

Moreover, the mean vector ψ_g is no longer equal to the Shapley value ϕ_h, as shown by the next lemma.

Lemma 5. *For a game (V, h) with an intersecting supermodular and nega-modular function $h : 2^V \mapsto \Re_+$, there is no algorithm that computes the Shapley value $\phi_h \in \Re^V$ by using less than $2^n - 1$ function value oracle calls, where $n = |V|$.* $\square$

4 Conclusion

In this paper, we showed that, for an intersecting submodular and posi-modular set function f on V, its polyhedron $P_-(f)$ is described by a set of inequalities $z(X) \leq f(X)$ such that X is in a laminar family $\mathcal{X} \subseteq 2^V$. Furthermore, such a laminar family can be obtained combinatorially by $O(|V|^3)$ function value oracle calls. This significantly reduces the complexity of finding a vector z in the polyhedron $P_-(f)$. As a result, we show that several optimization problems over the polyhedron have efficient combinatorial algorithms, and that the core and its mean vector of some cooperative game can be efficiently computed. It is left for the future research to widen the class of set functions to which similar algorithms are applicable.

References

1. G.-R. Cai and Y.-G. Sun, *The minimum augmentation of any graph to k-edge-connected graph*, Networks, 19, 1989, 151–172. 170, 171
2. J. Edmonds, *Submodular functions, matroids, and certain polyhedra*, Proc. Calgary Int. Conference on Combinatorial Structures and Their Applications (R. Guy, H. Hanani, N. Sauer and J. Schönheim, eds., Gordon and Breach, New York), 1970, 69–87. 177
3. A. Frank, *Augmenting graphs to meet edge-connectivity requirements*, SIAM J. Discrete Mathematics, 5, 1992, 25–53. 170, 171
4. A. Frank, *Applications of submodular functions*, in Surveys in Combinatorics, Keith Walker, ed., London Math. Soc. Lecture Notes Ser. 187, 1993, 85–136.
5. T. Ibaraki and N. Katoh, Resource Allocation Problems – Algorithmic Approaches, Foundations of Computing Series, MIT Press, 1988. 170
6. H. Nagamochi and T. Ibaraki, *A note on minimizing submodular functions*, Information Processing Letters (to appear). 170, 172
7. L. S. Shapley, *A value for n-person games*, H. W. Kuhn and A. W. Tucker (eds), Contributions to the Theory of Games, II, Annals of Mathematics Studies, 28, 1953, 307–317. 177
8. L. S. Shapley, *Cores of convex games*, Int. J. of Game Theory, 1, 1971, 11–26. 170, 177

Maximizing the Number of Connections in Optical Tree Networks

Thomas Erlebach[⋆,1] and Klaus Jansen[⋆⋆,2]

[1] TU München, 80290 München, Germany
erlebach@in.tum.de
[2] IDSIA Lugano, Corso Elvezia 36, 6900 Lugano, Switzerland
klaus@idsia.ch

Abstract. In optical networks with wavelength division multiplexing (WDM), multiple connections can share a link if they are transmitted on different wavelengths. We study the problem of satisfying a maximum number of connection requests in a directed tree network if only a limited number W of wavelengths are available. In optical networks without wavelength converters this is the *maximum path coloring* (MaxPC) problem, in networks with full wavelength conversion this is the *maximum path packing* (MaxPP) problem. MaxPC and MaxPP are shown to be polynomial-time solvable to optimality if the tree has height one or if both W and the degree of the tree are bounded by a constant. If either W or the degree of the tree is not bounded by a constant, MaxPC and MaxPP are proved $\mathcal{NP}$-hard. Polynomial-time approximation algorithms with performance ratio $5/3 + \varepsilon$ for arbitrarily small ε are presented for the case $W = 1$, in which MaxPC and MaxPP are equivalent. For arbitrary W, a 2-approximation for MaxPP in arbitrary trees, a 1.58-approximation for MaxPC in trees of bounded degree, and a 2.22-approximation for MaxPC in arbitrary trees are obtained.

1 Introduction

All-optical communication networks are the technology of choice for satisfying the ever-growing demands for telecommunication bandwidth. Data is transmitted through optical fiber at the rate of gigabits-per-second, and WDM (wavelength-division multiplexing) allows several connections to use a link simultaneously if the signals are transmitted on different wavelengths. In all-optical networks with switches without wavelength conversion capabilities, a connection must use the same wavelength on the whole path from transmitter to receiver. If W wavelengths are available, a set of connections can be established if each connection is assigned a transmitter-receiver path and one of the W wavelengths such that connections sharing a directed link receive different wavelengths. Another model of optical networks employs switches with wavelength conversion. A

⋆ supported by DFG contract SFB 342 TP A7
⋆⋆ supported partially by the EU ESPRIT LTR Project NO. 20244 (ALCOM-IT)

connection can use different wavelengths on different segments of its transmitter-receiver path. If all network switches have full wavelength conversion capabilities, a set of connections can be established if they are assigned transmitter-receiver paths such that no directed link is used by more connections than the number of available wavelengths.

As the number of distinct available wavelengths is small in practice, it is not always the case that all requests in a given set of connection requests can be established simultaneously under the constraints mentioned above. In such a scenario, the network provider might decide to reject some requests and to maximize the number of accepted requests. We investigate the complexity and approximability of this optimization problem for the case that the topology of the network is that of a *directed tree*, i.e., the graph obtained from an undirected tree by replacing each undirected edge by two directed edges with opposite directions.

1.1 Preliminaries

A *connection request* in a directed tree T is given by a sender-receiver pair (u, w) and corresponds to the directed path from u to w in T. We will refer to requests as paths in the remainder of this paper. Two paths *intersect* if they share a directed edge of T. For a given set P of paths, the *load* $L(e)$ of a directed edge e of T is the number of paths in P using edge e, and L denotes the maximum load among all edges of T. A W-coloring of a given set of paths is an assignment of colors (wavelengths) to the paths using at most W colors such that intersecting paths receive different colors.

For given directed tree T, set P of paths in T, and number W of available colors, the *maximum path coloring* problem (MaxPC) is to compute a subset $P' \subseteq P$ and a W-coloring of P' such that $|P'|$ is maximized, and the *maximum path packing* problem (MaxPP) is to compute a subset $P' \subseteq P$ with maximum load W such that $|P'|$ is maximized. For a given instance of MaxPC or MaxPP, we denote by P^* an arbitrary optimum solution. MaxPC models optical networks without wavelength converters, while MaxPP models the availability of full wavelength conversion.

By Δ we denote the maximum outdegree of all nodes in the given directed tree T. We assume that an arbitrary node r of T has been designated the root of T. For $v \neq r$, $p(v)$ denotes the parent of v. The *level* of a node is its distance (number of edges) from r. For a pair (u, w) of nodes in T, we denote by $\mathrm{lca}(u, w)$ the unique *least common ancestor* (lca) of u and w, i.e., the node with smallest level among all nodes on the path from u to w. The one or two edges on a path (u, w) that are incident to $\mathrm{lca}(u, w)$ are called the *top edges* of the path (u, w). A subtree of T *contains* a path (u, w) if $\mathrm{lca}(u, w)$ is a node of the subtree.

A polynomial-time algorithm is a ρ-approximation algorithm for MaxPC or MaxPP if it always outputs a set P' whose cardinality is at least a $(1/\rho)$-fraction of the cardinality of an optimum solution. An algorithm that computes an optimum solution in polynomial time is called an *exact* algorithm.

1.2 Related Work

Previous work has focused on the *path coloring problem*, where the goal is to assign wavelengths to all given connection requests while minimizing the number of different wavelengths used. For undirected trees, a (3/2)-approximation algorithm was given in [10] and improved to an asymptotic 1.1-approximation in [2]. For directed trees, the best known algorithm colors a given set of directed paths with maximum load L using at most $\lceil (5/3)L \rceil$ colors [8,5,7,6]. While the path coloring problem is relevant when a provider designs a network in order to meet the given demands or when the network has enough capacity for satisfying all given requests, MaxPC and MaxPP apply to the case where an existing network has insufficient capacity and the goal is to maximize the number of accepted requests.

In chain networks, MaxPC and MaxPP are equivalent and can both be solved optimally in polynomial time by finding a maximum W-colorable subgraph in the conflict graph, which is an interval graph in this case [12]. For undirected trees, the exact algorithm for integral multicommodity flow with unit edge capacities from [3] (see below) gives an exact algorithm for MaxPC and MaxPP with $W = 1$, and using the reduction from Sect. 4 a 1.58-approximation for MaxPC with arbitrary W is obtained. The same results can be derived for ring networks [11]. For ring networks with predetermined routing of the given requests, a 3/2-approximation algorithm for MaxPC was obtained in [9]. A variant of the MaxPC problem is considered in [1]; see Sect. 4 for more details.

MaxPP is closely related to the integral multicommodity flow problem. Integral multicommodity flow and multicut have been studied for undirected trees in [3]. For a given set of source-sink pairs (commodities) in an undirected tree with edge capacities, the integral multicommodity flow problem is to maximize the sum of the flows of the commodities constrained by the given edge capacities. Exact algorithms for integral multicommodity flow are obtained in [3] for trees of height one with arbitrary edge capacities and for arbitrary trees with unit edge capacities. For trees with edge capacities 1 or 2 the problem is proved $\mathcal{NP}$-hard and MAX SNP-hard. For trees with arbitrary edge capacities, a 2-approximation algorithm is given. The main differences between the multicommodity flow problem considered in [3] and the MaxPP problem studied in the present paper are that we investigate directed instead of undirected trees and that no commodity can have a flow greater than 1 in our setting.

1.3 Results

We determine the complexity of MaxPP and MaxPC in directed trees under various restrictions and give approximation algorithms with small constant approximation ratios for the variants that are $\mathcal{NP}$-hard. We are not aware of any previous results regarding MaxPP and MaxPC in directed trees. Our complexity results are listed in this section without giving proofs; details can be found in the full paper available from the authors.

MaxPP and MaxPC are equivalent if the given tree has height one and can be solved in polynomial time using an algorithm for capacitated b-matching [4, pp. 257–259]. This algorithm extends to the weighted version of MaxPC and MaxPP, where each path p has associated benefit $b(p)$ and the goal is to maximize the total benefit of accepted paths. MaxPP and MaxPC can also be solved optimally in polynomial time using a bottom-up computation if the maximum degree Δ of the given tree network and the number W of available wavelengths are bounded by a constant. This algorithm extends to the weighted version of MaxPC and MaxPP as well, and also to variants where the set of available wavelengths can vary from link to link or where wavelength converters with limited conversion are allowed. (Furthermore, variants of the algorithm give exact algorithms for integral multicommodity flow in directed or undirected trees of bounded degree, if the edge capacities are bounded by a constant.) If either Δ or W can be arbitrarily large, MaxPP and MaxPC become $\mathcal{NP}$-hard. More precisely, both problems are $\mathcal{NP}$-hard for $W = 1$ and arbitrary degree, and for arbitrary W and degree bounded by 3 (i.e., binary trees).

For MaxPP with arbitrary W, we adapt the algorithm from [3] and obtain a 2-approximation (Sect. 2). If $W = 1$, i.e., only one wavelength is available, then MaxPC and MaxPP are equivalent to finding a maximum cardinality subset of edge-disjoint paths, and we give, as our main result, a family of polynomial-time approximation algorithms with approximation ratio $5/3+\varepsilon$ for this case, where ε can be chosen arbitrarily small (Sect. 3). For MaxPC with arbitrary W, we obtain a 2.22-approximation for trees of arbitrary degree and a 1.58-approximation for trees whose degree is bounded by a constant (Sect. 4).

2 Approximating MaxPP

The algorithm is as follows. Initially, set $P' = \emptyset$. Then process all nodes of the tree in order of non-increasing levels. When processing node v, consider the paths whose lca is v in arbitrary order. Insert each such path in P' if this does not increase the maximum load of P' above W. In the end, output P'. (Note that this algorithm for MaxPP works also if the number of available wavelengths is different on different links.)

Theorem 1. *The algorithm is a 2-approximation algorithm for MaxPP, i.e., it outputs a subset P' of P with maximum load at most W and with cardinality at least $|P^*|/2$.*

Proof. First, we observe that the approximation algorithm from [3] works also for directed trees. The only further difference between the multicommodity flow problem in [3] and the MaxPP problem is that, with the MaxPP problem, no commodity can have flow greater than 1. But our greedy algorithm for MaxPP in a directed tree T with edge capacity W behaves like the algorithm from [3] in a slightly extended tree T': for each path (commodity) p from a node u to a node w, add two new nodes u_p and w_p, add two unit capacity edges (u_p, u) and (w, w_p), and replace p by a path from u_p to w_p. The multicommodity flow problem in the resulting tree T' is equivalent to the MaxPP problem in the original tree, and our greedy algorithm produces the same solution as the 2-approximation algorithm from [3] on this instance. $\square$

3 A Family of Approximation Algorithms for $W = 1$

The algorithm from the previous section achieves approximation ratio 2 also for $W = 1$. The main idea that leads to an improvement in this special case is to consider all paths with the same lca simultaneously instead of one by one. This way we obtain, for any fixed $\varepsilon > 0$, a polynomial-time $(5/3 + \varepsilon)$-approximation algorithm.

Let P_v denote the subset of all paths $(u, w) \in P$ with $\mathrm{lca}(u, w) = v$ that do not intersect any of the paths that have been accepted by the algorithm at a previous node and that do not use any edges that have been *reserved* or *fixed* by the algorithm (see below). We assume without loss of generality that we have $u \neq v \neq w$ for all paths $(u, w) \in P_v$.

During a first pass, the algorithm processes the nodes of T in order of non-increasing levels. When the algorithm processes node v, it tries to determine for the paths in P_v whether they should be included in P' (these paths are called *accepted*) or not (these paths are called *rejected*). But the algorithm can not always make this decision right away. In some cases it leaves some paths in an intermediate state and resolves them later on. The possibilities for paths in such intermediate states are *undetermined paths*, *groups of deferred paths*, *groups of exclusive paths*, and *groups of 2-exclusive paths*. We refer to undetermined paths, groups of exclusive paths, and groups of 2-exclusive paths (but not groups of deferred paths) as *unresolved* paths.

If all paths in P_v use the same two top edges, accepting one of them might cause the algorithm to miss the chance of accepting two paths with an lca of smaller level later on. Instead, the algorithm picks one of the paths in P_v and makes it an *undetermined path* in this case.

Sometimes the algorithm decides to accept one of several intersecting paths, but it defers the decision which one of them to accept. The intersecting paths are called a *group of deferred paths* (see Fig. 1), and some edges (indicated by dotted arrows in Fig. 1) are marked as *reserved*. The motivation for introducing groups of deferred paths is as follows: first, the reserved edges block at most one path with lca of smaller level that could be accepted in an optimum solution; second, no matter which paths with lca of smaller level not intersecting a reserved edge are accepted by the algorithm later on, there is still at least one deferred path that can be accepted in a second pass that proceeds top-down.

A *group of exclusive paths* is sketched in Fig. 2(a). Such a group consists of a *lower path* p and a *higher path* q with lca of smaller level that intersects p. At most one of the two paths can be accepted, but the algorithm can not afford to pick the wrong one. It only marks the top edge of p that is intersected by q as *fixed* (indicated by a dotted arrow in Fig. 2). Groups of exclusive paths have the following property.

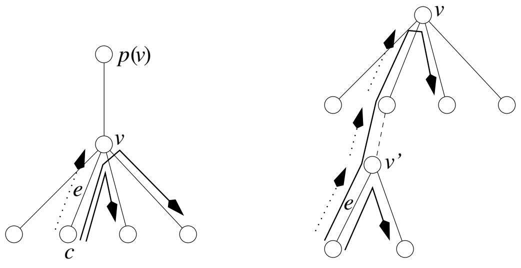


Fig. 1. Possible configurations of deferred paths

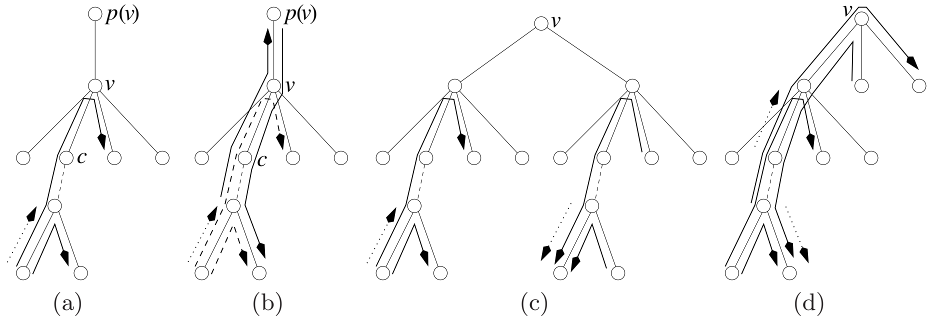


Fig. 2. (a) Possible configuration of exclusive paths; (b) situation in which both exclusive paths are blocked; (c) and (d) groups of 2-exclusive paths

Property (E): *As long as at most one path touching v but not using the fixed edge is accepted at a later node, either p or q can still be accepted. Only when two paths touching v are accepted at a later node, they may block both p and q from being accepted (see Fig. 2(b)).*

The last types of unresolved paths are sketched in Fig. 2(c) and (d). These *groups of 2-exclusive paths* consist of a set of four paths at most two of which can be accepted and have the following property.

Property (2E): *If at most one path touching v but not using a fixed edge is accepted at a later node, two paths from the group of 2-exclusive paths can still be accepted. If two paths touching v but not using a fixed edge are accepted at a later node, at least one path from the group of 2-exclusive paths can still be accepted.*

When the algorithm has finished processing a node v, the subtree rooted at v will contain at most one of the following: one undetermined path, or one group of exclusive paths, or one group of 2-exclusive paths. All other paths in the subtree are accepted, rejected, or member of a group of deferred paths.

3.1 Invariants

In the next subsection we will sketch how the algorithm proceeds during the first pass. At the same time, we will show that the approximation ratio achieved by the algorithm is $5/3+\varepsilon$. For establishing this, it can be proved by induction that the following invariant can be maintained. This invariant holds before the first node of T is processed, and it holds again each time an additional node of T has been processed.

Let A be the set of all paths that have already been accepted by the algorithm. Let F be the set of all paths in P whose lca has not yet been processed and which are not blocked by any of the accepted paths, by reserved edges of deferred paths, or by fixed edges. Let d be the number of groups of deferred paths that are contained in processed subtrees. Let U be the set of all undetermined paths. Let X be the union of all groups of exclusive paths and groups of 2-exclusive paths. Then there is a set $O \subseteq F \cup U \cup X$ of edge-disjoint paths satisfying the following conditions:
Condition 1: $|P^*| - |O| \leq (5/3+\varepsilon)(|A|+d)$
Condition 2: *For every group of exclusive paths, O contains one path from that group; for every group of 2-exclusive paths, O contains two paths from that group.*

O represents a set of paths that could still be accepted by the algorithm and that has the following property: if the algorithm accepts at least a $1/(5/3+\varepsilon)$-fraction of the paths in O (in addition to the paths it has already accepted), its output is a $(5/3+\varepsilon)$-approximation of the optimum solution.

The invariant is satisfied initially with $A=\emptyset$, $d=0$, $F=P$, $U=\emptyset$, $X=\emptyset$, and $O=P^*$. In order to prove that the invariant can be maintained, it suffices to show how the set O that establishes the invariant before node v is processed can be manipulated so as to satisfy the invariant also after node v is processed. In particular, some paths must be replaced in O or removed from O in order to satisfy $O \subseteq F \cup U \cup X$ and to keep O a set of edge-disjoint paths after v is processed, and it must be shown that the number of paths removed from O is at most $(5/3+\varepsilon)(a_v+d_v)$ if the algorithm accepts a_v additional paths and creates d_v new groups of deferred paths while processing v (thus satisfying Condition 1). Condition 2 must only be considered explicitly when a new group of exclusive paths or group of 2-exclusive paths is created.

If the invariant is satisfied after the root node is processed, we have $F=\emptyset$, $O \subseteq U \cup X$, and $|P^*| - |O| \leq (5/3+\varepsilon)(|A|+d)$. By accepting the undetermined path (if any), accepting an arbitrary path from the group of exclusive paths (if any), and accepting two arbitrary edge-disjoint paths from the group of 2-exclusive paths (if any), the algorithm accepts $|O|$ additional paths, and the resulting set A satisfies $|P^*| \leq (5/3+\varepsilon)(|A|+d)$. In the second pass, the algorithm processes the nodes of the tree in order of non-decreasing levels and accepts g additional paths at each node v that is the lca of g groups of deferred paths. Then the algorithm outputs the set of all accepted paths. As the algorithm accepts d additional paths, one from each group of deferred paths, in the second pass, this establishes our main theorem.

Theorem 2. *For every fixed $\varepsilon > 0$, there is a polynomial-time approximation algorithm for MaxPC and MaxPP with one available color (wavelength) having approximation ratio $5/3 + \varepsilon$.*

3.2 The First Pass

Recall that $P_v \subseteq P$ is the set of all paths with lca v that do not intersect any previously accepted path nor any fixed or reserved edge. Let U_v be the set of undetermined paths contained in subtrees rooted at children of v. Let X_v be the union of groups of exclusive paths and groups of 2-exclusive paths contained in subtrees rooted at children of v. We sketch how the algorithm processes node v and determines which of the paths in $P_v \cup U_v \cup X_v$ should be accepted, rejected, deferred, or left in an unresolved state.

Note that for a given set of paths with lca v the problem of determining a maximum subset of edge-disjoint paths is equivalent to finding a maximum matching in a bipartite graph [8] and can thus be done in polynomial time. Furthermore, we use the following property of bipartite graphs: for $s = 1$ or $s = 2$, the fact that a maximum matching in a bipartite graph G has cardinality s implies that there are s vertices in G such that every edge is incident to at least one of these s vertices. (The property holds for arbitrary s and is known as the König theorem.)

Let k be the number of children of v that have an undetermined path in their subtree, let ℓ be the number of children that have a group of exclusive paths, and let m be the number of children that have a group of 2-exclusive paths. We use the expression *subtrees with exclusive paths* to refer to all subtrees of v with either a group of exclusive paths or with a group of 2-exclusive paths. Before v is processed, the invariant implies that there is a set $O \subseteq F \cup U \cup X$ satisfying Condition 1 and 2. In each single case of the following case analysis, it must be shown how a set O' can be obtained from O such that Condition 1 and 2 are satisfied for O' after v is processed.

Case 1: $k+\ell+m \leq \max\{3, 2/\varepsilon\}$. The algorithm can try out all combinations of accepting or rejecting unresolved paths in the subtrees rooted at children of v: for undetermined paths there are two possibilities (accepting or rejecting the path), for groups of exclusive paths there are two possibilities (accepting the lower path or accepting the higher path), and for groups of 2-exclusive paths there are at most four relevant possibilities of accepting two edge-disjoint paths of the group (see Fig. 2). Hence, the number of possible combinations is bounded from above by $2^{k+\ell}4^m = O(1)$.

For each of these combinations, the algorithm can compute a maximum number of edge-disjoint paths in P_v not intersecting any of the (tentatively) accepted paths from $U_v \cup X_v$. Let s be the maximum, over all combinations, of the number of tentatively accepted paths from $U_v \cup X_v$ plus the number of maximum edge-disjoint paths in P_v. If $s = 0$, we have $k = \ell = m = 0$ and $P_v = \emptyset$, and the algorithm proceeds with the next node. Otherwise, we distinguish the following cases.

If $s \geq 3$, the algorithm accepts the s paths and rejects all other paths from $P_v \cup U_v \cup X_v$. As s is the maximum number of edge-disjoint paths in $P_v \cup U_v \cup X_v$, O can contain at most s paths from $P_v \cup U_v \cup X_v$. Furthermore, O can contain at most two paths from F using the edges $(v, p(v))$ or $(p(v), v)$, and these are the only two further paths in O that could possibly be blocked by the s paths accepted by the algorithm. Hence, a valid set O' can be obtained from O by deleting at most $s+2$ paths. As $s+2 \leq (5/3)s$, the invariant is maintained. In the cases $s = 1$ and $s = 2$, a huge number of subcases for k, ℓ and m such that $k+\ell+2m \leq 1$ resp. $k+\ell+2m \leq 2$ is distinguished. For each of these subcases, a number of configurations of paths in P_v with respect to the paths in $X_v \cup U_v$ are considered. Each such subcase and configuration can be recognized in polynomial time, and it can be shown that one of the following actions can satisfy the invariant: (1) The algorithm creates a new undetermined path or a new group of exclusive paths. A valid set O' can be derived from O by replacing or inserting one path, if necessary. (2) The algorithm creates a new group of deferred paths. In this case, a valid set O' can be derived from O by deleting at most one path. (3) The algorithm accepts paths and/or creates groups of deferred paths such that $a_v + d_v = 2$. In this case, a valid set O' can be derived from O by deleting at most 3 paths. (4) The algorithm creates a group of 2-exclusive paths from some paths in $P_v \cup X_v \cup U_v$. In this case, a valid set O' can be derived from O by replacing or inserting at most two paths. All details are omitted in this extended abstract due to space limitations.

Case 2: $k + \ell + m > \max\{3, 2/\varepsilon\}$. The algorithm calculates four candidate sets $S_1,\ldots,S_4$ of edge-disjoint paths from $P_v \cup U_v \cup X_v$ and chooses the largest of them. For obtaining S_2 and S_4, we employ a method of removing r paths from an arbitrary set S of edge-disjoint paths in P_v such that $\ell + 2m$ exclusive paths from X_v can be accepted in addition to the paths remaining in S. The details of the method and a proof that $r \leq (|S| + \ell + m)/3$ are omitted. S_1 is obtained as the union of all k undetermined paths and a maximum number s_1 of edge-disjoint paths from P_v not intersecting any undetermined path, and as many additional edge-disjoint paths from the $\ell + m$ subtrees with exclusive paths as possible. We have $|S_1| \geq k + s_1 + m$, because S_1 contains k undetermined paths and at least m paths from groups of 2-exclusive paths in X_v due to Property (2E). S_2 is obtained from S_1 by removing r of the s_1 paths in $S_1 \cap P_v$ from S_1 until $\ell+2m$ exclusive paths can be accepted. S_2 contains $\ell + 2m$ exclusive paths, and only $r \leq (s_1+\ell+m)/3$ of the s_1 paths in $S_1 \cap P_v$ were removed to obtain S_2. As S_2 still contains the k undetermined paths, we have $|S_2| \geq k+m+(2/3)(s_1+\ell+m)$. S_3 is obtained by taking a maximum number s_3 of edge-disjoint paths from P_v and as many additional edge-disjoint paths from the $\ell+m$ subtrees with exclusive paths and the k subtrees with undetermined paths as possible. We have $|S_3| \geq s_3 + m$, because S_3 contains at least m paths from groups of 2-exclusive paths in X_v due to Property (2E). S_4 is obtained from S_3 by removing r of the s_3 paths in $S_3 \cap P_v$ from S_3 until $\ell + 2m$ exclusive paths can be accepted. Since $r \leq (s_3 + \ell + m)/3$, we have $|S_4| \geq m + (2/3)(s_3 + \ell + m)$. We claim that the number of paths in $O_v = O \cap (P_v \cup U_v \cup X_v)$ is at most $s_1 + (s_3 - s_1)/2 + k + \ell + 2m$. With this upper bound on $|O_v|$ and the lower bounds on the cardinalities of the four sets S_i, it can be proved that at least one of the sets S_i satisfies $|O_v|+2 \leq (5/3+\varepsilon)|S_i|$. At most $|O_v|+2$ paths must be removed from O in order to obtain a valid set O'.

4 Approximating MaxPC for Arbitrary W

In order to obtain an approximation algorithm for MaxPC with arbitrary number W of available wavelengths from an algorithm for $W = 1$, we employ a technique from [1]. The approximation algorithm A for arbitrary number W of wavelengths is obtained from an approximation algorithm A_1 for one wavelength by running W copies of A_1 and giving as input to the i-th copy the set of paths that have not been accepted by the first $i-1$ copies of A_1. The output of A is the union of the W sets of paths output by the copies of A_1, and the paths in the i-th set are assigned colored i.

In [1] it is shown that the algorithm A obtained in this way has approximation ratio at most $\rho+1$ if A_1 has approximation ratio ρ, even if different wavelengths are associated with different network topologies. There the technique is used to obtain randomized on-line algorithms with logarithmic competitive ratio for networks shaped as rooted forests. For identical networks, which we have in our application, the approximation ratio achieved by A can even be bounded by $1/(1-(1-1/(\rho W))^W)$, which is smaller than $1/(1-\mathrm{e}^{-1/\rho})$ for all W. This bound is mentioned in a preliminary draft of the journal version of [1], which was kindly supplied to the authors by Adi Rosén. The bound can be proved easily by using the fact that, if A has selected p_k paths after running k copies of A_1, there is still a set of at least $(|P^*|-p_k)/W$ edge-disjoint paths among the remaining paths (this follows from a pigeonhole argument), and the next copy of A_1 accepts at least a $(1/\rho)$-fraction of them. As we have obtained an exact algorithm for MaxPC with $W = 1$ in bounded degree trees and $(5/3+\varepsilon)$-approximation algorithms for MaxPC with $W = 1$ in arbitrary trees, by employing the above technique we obtain approximation algorithms with ratio $1/(1-1/e) \approx 1.58$ for arbitrary W in bounded degree trees and with ratio ≈ 2.22 for arbitrary W in arbitrary trees.

5 Open Problems

It is an interesting open question whether the approximation ratios of our algorithms for directed trees can be improved. Other promising directions for future research include approximation algorithms for the weighted MaxPP and MaxPC problems in arbitrary trees and for the MaxPC problem with different sets of available wavelengths on different links. In addition, it would be very interesting to see whether techniques we used in the $(5/3+\varepsilon)$-approximation for MaxPC and MaxPP with $W = 1$ can lead to improved approximation algorithms for the integral multicommodity flow problem in trees in general or for special cases thereof; for this problem, the best known approximation is still the 2-approximation from [3].

Acknowledgments

We are grateful to Stefano Leonardi for pointing out the reduction from MaxPC with arbitrary number of wavelengths to MaxPC with one wavelength, and to Adi Rosén for informing us about the improved analysis for the ratio obtained by this reduction in the case of identical networks for all wavelengths.

References

1. B. Awerbuch, Y. Azar, A. Fiat, S. Leonardi, and A. Rosén. On-line competitive algorithms for call admission in optical networks. In *Proceedings of the 4th Annual European Symposium on Algorithms ESA '96*, LNCS 1136, pages 431–444. Springer-Verlag, 1996. 181, 188
2. T. Erlebach and K. Jansen. Scheduling of virtual connections in fast networks. In *Proceedings of the 4th Parallel Systems and Algorithms Workshop PASA '96*, pages 13–32. World Scientific Publishing, 1997. 181
3. N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees, with applications to matching and set cover. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming, ICALP '93*, LNCS 700, pages 64–75. Springer-Verlag, 1993. 181, 182, 183, 188
4. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, Berlin, 1988. 182
5. C. Kaklamanis and P. Persiano. Efficient wavelength routing on directed fiber trees. In *Proceedings of the 4th Annual European Symposium on Algorithms ESA '96*, LNCS 1136, pages 460–470. Springer-Verlag, 1996. 181
6. C. Kaklamanis, P. Persiano, T. Erlebach, and K. Jansen. Constrained bipartite edge coloring with applications to wavelength routing. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming ICALP '97*, LNCS 1256, pages 493–504. Springer-Verlag, 1997. 181
7. V. Kumar and E. J. Schwabe. Improved access to optical bandwidth in trees. In *Proceedings of the 8th Annual ACM–SIAM Symposium on Discrete Algorithms SODA '97*, pages 437–444, 1997. 181
8. M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 548–557, 1995. 181, 186
9. C. Nomikos and S. Zachos. Coloring a maximum number of paths in a graph. Presented at the *Workshop on Algorithmic Aspects of Communication (July 11–12, 1997, Bologna, Italy)*. 181
10. P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing STOC '94*, pages 134–143, New York, 1994. ACM SIGACT, ACM Press. 181
11. P.-J. Wan and L. Liu. Maximal throughput in optical networks, 1998. Presented at the *DIMACS Workshop on Optical Networks (March 16–19, 1998)*. 181
12. M. Yannakakis and F. Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Inf. Process. Lett.*, 24(2):133–137, January 1987. 181

Selecting the k Largest Elements with Parity Tests

Tak Wah Lam and Hing Fung Ting

Department of Computer Science, University of Hong Kong, Hong Kong
{twlam,hfting}@csis.hku.hk

Abstract. In this paper we study the problem of finding the k largest elements of n distinct real numbers. We give a pure combinatorial argument to prove that $n + (k-1)\log n + O(1)$ queries are necessary for any deterministic algorithm using parity tests only. This settles an open problem first raised by Yao [11]. We also present a randomized algorithm with error probability $O(1/n^c)$ using only $O(\log^2 n + k \log n)$ parity tests, where $c > 1$ is any fixed constant.

1 Introduction

This paper is concerned with the problem of finding the k largest elements of n distinct real numbers. This problem, as well as other variants of the selection problem, has been studied extensively. Based on the pairwise comparison model (i.e., each query takes the form "$x_i - x_j : 0$" where x_i, x_j are distinct inputs), it is known that $n + (k-1)\log n + O(1)$ comparisons[1] are necessary and sufficient to find the k largest elements (Hyafil [4], Kirkpatrick [5], Kislitsyn [6], Pratt and Yao [7]). To go further, people have studied more general queries. Fussenegger and Gabow [2] showed that even when a query is any linear function of the inputs, the same lower bound applies. Yao [11] is the first to go beyond linear function and proved that the same lower bound holds for median test, which is of the form "$(x_i - x_j)(x_i - x_k) : 0$". Fleischer [1] extended this lower bound result to any arbitrary products of linear functions.

In this paper we focus on parity test, which is of the form "$\prod_{(i,j)\in C}(x_i - x_j) : 0$" where C is any collection of pairs of input positions. The study of parity test on selection problems is not new in the literature; it is actually the only kind of polynomial test people know how to make use of in deriving better upper bound results for selection problems (see e.g., [8,9]). Fleischer's result [1], of course, implies the above-mentioned lower bound for parity test. Nevertheless, the work of Yao [11] and Fleischer [1] for non-linear test is not straightforward (though thought-provoking); the key lemmas are based on geometric arguments, reasoning the relationship of the regions in the n-dimensional space defined by the queries. An open problem, first raised in [11], is to find a pure combinatorial proof of the lower bound for median test. This paper presents a simple combinatorial proof for parity test, settling this open problem.

[1] Throughout the paper, the base of the logarithm is 2.

The second contribution of this paper is an efficient randomized algorithm for finding the k largest elements. We show that, for any constant $c > 1$, there is a randomized algorithm with error probability $O(1/n^c)$ for finding the k largest elements using only $O(\log^2 n + k \log n)$ parity tests. We remark that randomization cannot help if only pairwise comparisons are allowed. More precisely, we show that the error probability of any randomized algorithm using $o(n)$ pairwise comparisons is at least $1 - o(1)$. In other words, such an algorithm returns an incorrect answer almost surely.

2 A Lower Bound for Selecting the *k* Largest Elements

In this section we show that if only parity tests are allowed, any deterministic algorithm for finding the k largest elements of n distinct numbers has to make at least $n - k + \lceil \log n(n-1) \cdots (n-k+1) \rceil$ tests in the worst case. An ordered pair (i, j) is said to be an *index pair* if i, j are distinct integers in $\{1, 2, \cdots, n\}$. A parity test takes the form of $\prod_{(i,j) \in C} (x_i - x_j) : 0$, where C is a set of index pairs.

2.1 Model of Computation

We model a selection algorithm by a decision tree. A decision tree is a binary tree in which every internal node is labeled with a test over the inputs (e.g., a parity test "$\prod_{(i,j)\in C}(x_i - x_j) : 0$"), and every leaf is labeled with an output (say, the indices of the largest elements). Every internal node has exactly two outgoing edges labeled with "> 0" and "< 0". Given any input $X = \{x_1, x_2, \cdots, x_n\}$, the execution of the selection algorithm corresponds to tracing a path from the root of the decision tree to a leaf. More precisely, starting from the root, we repeatedly perform the test specified in a node and follow the outgoing edge whose label matches the result of the test. The execution terminates when a leaf is reached. The output for the input X is the label of that leaf. The length of the longest path from the root of a decision tree to any of its leaf represents the number of tests the selection algorithm performs in the worst case.

2.2 A Counting Argument of Fussenegger and Gabow

Most of the existing lower bound results [1,2,3,11] on the problem of selecting the k largest elements make use of a counting argument introduced by Fussenegger and Gabow [2]:

> Given a decision tree T for finding the k largest elements of n numbers, it is always possible to construct $n(n-1) \cdots (n-k+2)$ subtrees of T which do not share any leaves and each of which defines an algorithm for finding the maximum of $n - k + 1$ numbers.

Assume that T only uses tests of type Q. If one can show that any decision tree that uses tests of type Q to find the maximum of m numbers must have at least 2^{m-1} leaves, then T must have $2^{n-k}n(n-1)\cdots(n-k+2)$ leaves, and its height must be at least $n-k+\lceil\log n(n-1)\cdots(n-k+2)\rceil$. The latter is also a lower bound on the number of tests made by T in the worst case.

In the next subsection, we give a proof of the following lemma.

Lemma 1. *If only parity tests are allowed, then any decision tree for finding the maximum element of n distinct numbers must have at least 2^{n-1} leaves.*

Lemma 1 together the above-mentioned counting argument give us the following theorem.

Theorem 1. *If only parity tests are allowed, then any algorithm for finding the k largest elements of n distinct numbers must make at least $n-k+\lceil\log n(n-1)\cdots(n-k+2)\rceil = n+(k-1)\log n+O(1)$ tests in the worst case.*

2.3 A Combinatorial Lower Bound Proof

Let T be any decision tree for finding the maximum element of n distinct real numbers using parity tests only. We assume that for any leaf of T, there exists at least one input $X=(x_1,x_2,\cdots,x_n)$ such that the execution for X terminates at this leaf. In this subsection we show that every path from the root to a leaf of T has at least $n-1$ internal nodes. Then Lemma 1 follows.

Pick an arbitrary leaf σ of T. Let h be the number of internal nodes along the path between the root and σ. Let $C_1,C_2,\cdots,C_h$ be the sets of index pairs corresponding to the tests at these internal nodes. Denote $\mathcal{G}$ as the collection $\{C_1,C_2,\cdots,C_h\}$. Without loss of generality, we make the following assumptions:

- σ is labeled with the output 1, i.e., the first input is the maximum.
- All the edges along the path from the root to σ are labeled with ">0". That is, if the execution for an input X terminates at σ, then $\prod_{(i,j)\in C_l}(x_i-x_j)>0$ for all $C_l\in\mathcal{G}$.

Suppose, by way of contradiction, that $\mathcal{G}$ contains $h<n-1$ sets of index pairs. Let $X=(x_1,x_2,\cdots,x_n)$ be an input whose execution terminates at σ. That is, x_1 is the maximum element in X. We are going to construct another input $Z=(z_1,z_2,\cdots,z_n)$ from X such that the execution for Z also terminates at σ, but z_1 is not the maximum element in Z. Thus, we obtain a contradiction.

To ease our discussion, we further assume that $C_1,C_2,\cdots,C_h$ are each *term-wise positive* with respect to X, i.e., for every index pair $(i,j)\in C_l$, $(x_i-x_j)>0$. Below we explain why this assumption is valid. Consider any $C_l\in\mathcal{G}$. Since $\prod_{(i,j)\in C_l}(x_i-x_j)>0$, there are an even number (possibly zero) of index pairs $(i,j)\in C_l$ such that $(x_i-x_j)<0$. Let us replace all such index pairs (i,j) with (j,i). Then C_l results in a test which produces the same outcome for all possible inputs and which is term-wise positive with respect to X.

Roughly speaking, if h (i.e. $|\mathcal{G}|$) is less than $n-1$, we can show that some x_i's are not tested thoroughly by $\mathcal{G}$. More specifically, we say that $U\subseteq\{1,2,\ldots,n\}$ is a set of obscure positions if U satisfies the following properties:

- U includes 1 and at least one other position in $\{2, \cdots, n\}$.
- For each $C_l \in \mathcal{G}$, there is an even number of index pairs (i, j) in C_l where i is any index and j is in U (i.e. $|\{(i,j) \in C_l \mid j \in U\}|$ is even).

Z will be constructed in such a way that the maximum element can be placed in any position in U other than the first position.

Before proving the existence of such a set U, we need one more notation. For any $C_l \in \mathcal{G}$ and $j \in \{1, 2, \cdots, n\}$, define N_{lj} to be the number of index pairs (i, j) in C_l, where i is any index in $\{1, 2, \cdots, n\}$. Obviously, $\sum_{j \in U} N_{lj} = |\{(i,j) \in C_l \mid j \in U\}|$.

Lemma 2. *For any $C_l \in \mathcal{G}$, $N_{l1} = 0$.*

Proof. Since every $C_l \in \mathcal{G}$ is term-wise positive with respect to X and x_1 is the maximum element of X, no C_l contains an index pair $(i, 1)$ for any $i \in \{1, 2, \cdots, n\}$. Thus, $N_{l1} = 0$.

Lemma 3. *Assume that $|\mathcal{G}| = h \leq n - 2$. There exists a nonempty set $U_1 \subseteq \{2, 3, \cdots, n\}$ such that for all $C_l \in \mathcal{G}$, $(\sum_{j \in U_1} N_{lj})$ is even.*

Proof. There are $2^{n-1} - 1$ distinct nonempty subsets of $\{2, 3, \cdots, n\}$. Let V be one of such subsets. For each $l \in \{1, \cdots, h\}$, $\sum_{j \in V} N_{lj}$ is either odd or even. We consider the parity of $\sum_{j \in V} N_{lj}$ for all l as a vector of length h. Since there are $h \leq n - 2$ C_l's, there are at most 2^{n-2} distinct vectors. Thus, there must exist two distinct subsets V_1 and V_2 such that for all $C_l \in \mathcal{G}$, $\sum_{j \in V_1} N_{lj}$ and $\sum_{j \in V_2} N_{lj}$ have the same parity.

Let U_1 be the symmetric difference of V_1 and V_2 (i.e. $(V_1 \cup V_2) - (V_1 \cap V_2)$). U_1 is non-empty. Moreover, for any $C_l \in \mathcal{G}$, $(\sum_{j \in U_1} N_{lj})$ is even.

By Lemmas 2 and 3, we can conclude that $U = \{1\} \cup U_1$ is a set of obscure positions. The next lemma shows how to construct an input Z based on such a set U, which contradicts that inputs terminates at σ should have their maximum element in the first position.

Lemma 4. *If $h \leq n-2$, then there is an input $Z = (z_1, z_2, \cdots, z_n)$ such that z_1 is not the maximum element and, for any $C_l \in \mathcal{G}$, $\prod_{(i,j) \in C_l} (z_i - z_j) > 0$.*

Proof. Let U be the set of obscure positions defined above. Recall that x_1 is the maximum element of X. Define Z follows. For any $i \in \{1, 2, \cdots, n\}$,

$$z_i = \begin{cases} -x_i & \text{if } i \in U, \\ x_i - 2x_1 & \text{otherwise.} \end{cases}$$

Consider any C_l in $\mathcal{G}$. C_l is term-wise positive with respect to X. Thus, for any index-pair $(i, j) \in C_l$, $(z_i - z_j) < 0$ if and only if $j \in U$. By definition of U, C has an even number of index pairs (i, j) where $j \in U$. We conclude that $\prod_{(i,j) \in C_l} (z_i - z_j) > 0$.

We claim that z_1 is not the maximum element in Z. Since $|U| \geq 2$, there exists $j_0 \in U$ and $j_0 \neq 1$. By definition of Z, $z_{j_0} = -x_{j_0}$ and $z_1 = -x_1$ Therefore, z_1 is smaller than z_{j_0} and z_1 is not the maximum element.

In conclusion, if the path from the root of T to σ contains $h < n-1$ parity tests, there exists an input Z of which the maximum element is not in the first position, yet the execution of T for Z terminates at σ. A contradiction occurs.

3 A Randomized Algorithm

In this section we present a randomized algorithm with error probability $O(n^{-c})$ for selecting the k largest elements of n distinct numbers $X = \{x_1, x_2, \ldots, x_n\}$ using $O(\log^2 n + k \log n)$ parity tests, where $c > 1$ is any fixed constant. The parity tests performed by our algorithm are actually very *restricted*, all in the form "$\prod_{j \in B}(x_i - x_j) : 0$" where B is any set of input positions. Our work is motivated by Ting and Yao's randomized algorithm [9] for finding the maximum of n numbers, which uses $O(\log^2 n)$ restricted parity tests and has error probability $O(n^{-c})$. Note that by repeating Ting and Yao's algorithm k times, we can select the k largest using $O(k \log^2 n)$ restricted parity tests, the error probability is $O(kn^{-c})$.

Our improved algorithm is called `Select`, which repeatedly calls a function called `FindLarger` to find an element in the input larger than a given element. The following notation will be used throughout this section. For any $I \subseteq \{1, 2, \cdots, n\}$, denote X_I as the subset $\{x_i \mid i \in I\}$.

3.1 `FindLarger`

`FindLarger` is a function which, given any $I \subseteq \{1, 2, \cdots, n\}$ and $s \in I$, finds an index $s' \in I$ such that $x_{s'} > x_s$. The function may fail to find such an s' and return 0. Details are as follows. We randomly choose a subset B of $I - \{s\}$ and perform the restricted parity test "$\prod_{j \in B}(x_s - x_j) : 0$". If $\prod_{j \in B}(x_s - x_j) < 0$, X_B must contain an odd number of elements greater than x_s. We find such an element using the following binary search. We randomly choose a subset of B with exactly $\lceil |B|/2 \rceil$ elements. Denote B' as this subset and let $B'' = B - B'$. We perform one more restricted parity test, $\prod_{j \in B'}(x_s - x_j) : 0$. Note that either $\prod_{j \in B'}(x_s - x_j)$ or $\prod_{j \in B''}(x_s - x_j)$ is less than zero. Depending on whether $\prod_{j \in B'}(x_s - x_j)$ or $\prod_{j \in B''}(x_s - x_j)$ is less than zero, we reduce the problem of finding an element in X_B that is greater than x_s to the same problem for B' or B'', respectively. By repeating the process $\lceil \log |B| \rceil$ times, we can eventually find an index s' with $x_{s'} > x_s$.

It remains to consider the case when $\prod_{j \in B}(x_s - x_j) > 0$. Since B may contain no elements greater than x_s, B is simply discarded (in this case, we say B fails). We choose another subset B randomly and repeat the process again. We will give up if, in $\lambda = \lceil c \log n + \log k \rceil$ attempts, the B's chosen all fail. In this case, `FindLarger` returns 0.

`FindLarger` makes at most $\lambda + \lceil \log n \rceil$ restricted parity tests. Note that all the random subsets used by `FindLarger` are chosen uniformly and independently. If `FindLarger` does report an index s' such that $x_{s'} > x_s$, then every index in the set $L = \{h \in I \mid x_h > x_s\}$, by symmetry, has equal probability to be reported. The following lemma is concerned with the probability that `FindLarger` successfully reports an index s'.

Lemma 5. *If x_s is the maximum element in X_I, then* `FindLarger` *returns* 0. *Otherwise, the function returns an index $s' \in L$ with probability $1 - \frac{1}{2^\lambda}$.*

Proof. If x_s is the maximum element in X_I, then $\prod_{j \in B}(x_s - x_j) > 0$ for any subset B of I. Thus, `FindLarger` must return 0. Assume that x_s is not the maximum element. For any subset B, the probability that B contains an even number of elements in L (and hence $\prod_{j \in B}(x_s - x_j) > 0$) is exactly 1/2. The probability that the subsets B chosen in the λ attempts all fail is $1/2^\lambda$. Thus, with probability $1 - \frac{1}{2^\lambda}$, `FindLarger` encounters a subset B such that $\prod_{j \in B}(x_s - x_j) < 0$. Then, using the binary search, we can find an index $s' \in B$ such that $x_{s'} > x_s$.

3.2 The Selection Algorithm

We are ready to describe the algorithm `Select`. The computation of `Select` is divided into k phases; the ith largest element of X is reported in Phase i. The major data structure kept by `Select` is a stack of indices, denoted S below. Initially, an element in $\{1, 2, \cdots, n\}$ is picked randomly and pushed onto S. In general, S stores the indices of some elements that have not yet been reported by `Select`.

The computation involved in Phase i is as follows. Let $x_{\ell_1}, x_{\ell_2}, \cdots, x_{\ell_{i-1}}$ be the elements reported in previous phases. Let $I = \{1, 2, \cdots, n\} - \{\ell_1, \ell_2, \ldots, \ell_{i-1}\}$. To find the ith largest element, `Select` attempts to determine the maximum element in X_I. The first contender is x_s, where s is the index at the top of S. `Select` invokes the function `FindLarger` to find out whether any element in X_I is greater than x_s.

`FindLarger`(I, s) returns 0: From Lemma 5, with probability at least $1 - \frac{1}{2^\lambda}$, x_s is the maximum element in X_I. `Select` declares that x_s is the ith largest element (though it may not be the case) and pops s out from S. If S becomes empty, an index in $I - \{s\}$ will be picked randomly and pushed onto S. `Select` then enters Phase $i + 1$.

`FindLarger`(I, s) returns an $s' > 0$: Because $x_{s'} > x_s$, we push s' onto S. Then $x_{s'}$ is the next contender for the ith largest element of X.

Note that the sequence of contenders being tested in each phase is strictly increasing. This implies that `Select` will exit a phase after calling `FindLarger` at most $n - 1$ times. However, we want to limit the total number of calls of `FindLarger` over all phases to be $O(\log n)$. Thus, `Select` keeps a counter *count* for recording the total number of times `FindLarger` have been called. Let $\eta = 7c \log n + 2k$. When *count* exceeds the value of η, `Select` gives up and simply returns "failure".

A formal description of the algorithm is given in Figure 1. `Select` calls `FindLarger` at most η times and each `FindLarger` makes at most $\lambda + \lceil \log n \rceil$

```
input: $X = (x_1, x_2, \ldots, x_n)$
output: The $k$ largest elements of $x$
  $I \leftarrow \{1, 2, \cdots, n\}$; $\eta = 10c\lceil \log n \rceil + 2k$;
  create an empty stack $S$;
  randomly pick an $s \in I$;
  push($S, s$); $phase \leftarrow 1$; $count \leftarrow 0$;
  while ($phase \leq k$) and ($count < \eta$) do {
      $s \leftarrow$ top($S$);
      $s' \leftarrow$ FindLarger($I, s$); $count \leftarrow count + 1$;
      if $s' \neq 0$ then push($S, s'$)
      else {
              output ($x_s$); $I \leftarrow I - \{s\}$;
              pop($S$);
              if $S = \phi$ then { randomly pick an index $z$ in $I$; push($S, z$); }
              $phase \leftarrow phase + 1$;
      }
  }       // end of while
  if $count = \eta$ then return ("failure");
```

Fig. 1. The Algorithm `Select`

restricted parity tests, the total number of restricted parity tests made by the algorithm is $O(\log^2 n + k \log n)$. Below, we estimate the error probability of `Select`. There are two possible causes of error: (1) At some phase (say, Phase i), `Select` determines the ith largest element of X incorrectly, and (2) `Select` returns "failure".

Lemma 6. *Let p be the probability that at some phase, say, Phase i,* `Select` *determines the ith largest element of X incorrectly. Then $p = O(n^{-c})$.*

Proof. Consider Phase i. Recall that I denotes the indices of elements not yet reported. `Select` determines incorrectly some x_s to be the ith largest element only if `Select` receives a 0 from the function call `FindLarger`(I, x_s), but x_s is not the maximum element in X_I. From Lemma 5, this occurs with probability $1/2^\lambda = 1/kn^c$. Since there are at most k phases, we have $p = O(n^{-c})$.

Next, we estimate the probability that `Select` returns "failure", or equivalently, the probability that $count = \eta$. Let m be the number of indices left in the stack S when `Select` terminates. Let Δ be the number of `pop` operations executed by `Select`. Since a `pop` operation is executed only after `Select` outputs some x_s, we have $\Delta \leq k$. The number of `push` operations executed by `Select` is $m + \Delta$. Every time after `Select` calls `FindLarger`, it performs a push and/or pop operation. Thus, $count \leq m + 2\Delta \leq m + 2k$.

$$\begin{aligned} \text{Prob}\,[count = \eta] &\leq \text{Prob}\,[m + 2k \geq 7c \log n + 2k] \\ &\leq \text{Prob}\,[m \geq 7c \log n] \end{aligned}$$

Lemma 7. *Prob* $[m \geq 7c \log n] = O(n^{-c})$.

Proof. Assume that when `Select` terminates, the sequence of indices left in the stack S is $(s_0, s_1, \ldots, s_{m-1})$, where s_{m-1} is at the top. Consider any $j \in \{0, 1, \cdots, m-1\}$. At the point just before s_j is computed, let I_j denote the set of indices of elements not yet reported by `Select`. By definition, $I_{m-1} \subseteq I_{m-2} \subseteq \cdots \subseteq I_0$. In fact, `Select` computes s_j by calling the function `FindLarger`(I_j, s_{j-1}). Thus, $s_{m-1} \geq s_{m-2} \geq \cdots \geq s_0$. Recall that `FindLarger` is constructed in such a way that every index in the set $L_j = \{h \in I_j \mid x_h \geq x_{s_{j-1}}\}$ has equal probability to be reported as s_j. We say that s_j makes a *big advance* if x_{s_j} is larger than half or more of the elements in X_{L_j}, otherwise a small advance. Obviously, the probability that s_j makes a small advance is at most $1/2$. Among $s_0, s_1, \ldots, s_{m-1}$, there are at most $\lfloor \log n \rfloor$ indices giving big advances; we denote m' as the number of indices making small advances. Intuitively, the probability that m is large is very small because many small advances must be involved, yet each has probability at most $1/2$.

$$\begin{aligned}
\text{Prob}\,[m \geq 7c \log n] &\leq \text{Prob}\,[m' \geq 7c \log n - \log n] \\
&\leq \binom{7c \log n}{7c \log n - \log n} \left(\frac{1}{2}\right)^{7c \log n - \log n} \\
&\leq \left(\frac{e \cdot 7c \log n}{\log n}\right)^{\log n} \left(\frac{1}{2}\right)^{7c \log n - \log n} \\
&= n^{\log(e \cdot 7c) - (7c-1)} \\
&\leq n^{-c} \quad (\text{assume } c > 1)
\end{aligned}$$

The following theorem follows from Lemmas 6 and 7.

Theorem 2. *The algorithm* `Select` *finds the k largest elements of n distinct real numbers with error probability $O(n^{-c})$.*

4 Remarks

If only pairwise comparisons are allowed, any randomized algorithm that finds the maximum of n numbers using $o(n)$ comparisons must have error probability $1 - o(1)$. By Yao's Minimax Principle [10], this follows from the following proposition.

Proposition 1. *Let A be any deterministic algorithm that finds the maximum of n numbers using $h = o(n)$ comparisons. For a random input X, the probability that A returns an incorrect answer is $1 - o(1)$.*

Proof. By making h comparisons, A can examine at most $2h$ elements of X. Each of the remaining $n - 2h$ elements can be the maximum element with probability $1/n$, and A can return only one of them. Thus, the probability that an incorrect answer is returned is at least $(n - 2h - 1)/n$, which is $1 - o(1)$ as $h = o(n)$.

References

1. R. Fleischer. Decision trees: old and new results. In *Proceedings of the 25th Annual ACM symposium on Theory on Computing*, pages 468–477, 1993. 189, 190
2. F. Fussenegger and H.N. Gabow. Using comparison trees to derive lower bounds for selection problems. *Journal of the ACM*, pages 227–238, 1979. 189, 190
3. W.I. Gasarch. On selecting the k largest with restricted quadratic queries. *Information Processing Letters*, pages 193–195, 1991. 190
4. L. Hyafil. Bounds for selection. *SIAM Journal on Computing*, pages 109–114, 1976. 189
5. D. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *Journal of the ACM*, pages 150–165, 1981. 189
6. S.S. Kislitsyn. On the selection of the k-th element of an ordered set by pairwise comparisons. *Sibirskiĭ Mat. Zhurnal*, pages 557–564, 1964. 189
7. V. Pratt and F. Yao. On lower bounds for computing the ith largest element. In *Proceedings of 14th Annual Symposium on Switching and Automata Theory*, pages 70–81, 1973. 189
8. E. Reingold. Computing the maxima and the median. In *Proceedings of the 12th Annual IEEE Symposium on Switching and Automata Theory*, pages 216–218, 1971. 189
9. H.F. Ting and A.C. Yao. A randomized algorithm for finding maximum with $o((\log n)^2)$ polynomial tests. *Information Processing Letters*, pages 39–43, 1994. 189, 193
10. A.C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977. 196
11. A.C. Yao. On selecting the k largest with median tests. *Algorithmica*, pages 293–299, 1989. 189, 190

Randomized K-Dimensional Binary Search Trees*

Amalia Duch[1], Vladimir Estivill-Castro[2], and Conrado Martínez[1]

[1] Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
E-08028, Catalonia, Spain
{duch,conrado}@lsi.upc.es.

[2] Department of Computer Science and Software Engineering
The University of Newcastle
Callaghan 2308, Australia
vlad@cs.newcastle.edu.au.

Abstract. We introduce randomized K-dimensional binary search trees (randomized Kd-trees), a variant of K-dimensional binary search trees that allows the efficient maintenance of multidimensional records for any sequence of insertions and deletions; and thus, is fully dynamic. We show that several types of associative queries are efficiently supported by randomized Kd-trees. In particular, a randomized Kd-tree with n records answers exact match queries in expected $\mathcal{O}(\log n)$ time. Partial match queries are answered in expected $\mathcal{O}(n^{1-f(s/K)})$ time, when s out of K attributes are specified (with $0 < f(s/K) < 1$ a real valued function of s/K). Nearest neighbor queries are answered on-line in expected $\mathcal{O}(\log n)$ time. Our randomized algorithms guarantee that their expected time bounds hold irrespective of the order and number of insertions and deletions.

1 Introduction

The associative retrieval problem consists of answering queries with respect to a file of multidimensional records. Each multidimensional record contains a K-tuple of values that constitute its K-dimensional key and associated data. The entries of a K-dimensional key are referred to as *attributes*. An *associative query* specifies a certain condition about the keys and requires the retrieval of records in the file whose keys satisfy the given condition (examples of associative queries are intersection queries and nearest neighbor queries).

Several data structures for building information retrieval systems support associative queries and offer different trade-offs to the type of associative queries for which they are efficient [6,12]. Their space requirements, worst-case and expected-case performance on a range of operations as well as their design make

* This work is supported by the CONACyT grant # 89422 and by the ESPRIT-LTR Project ALCOM-IT contract # 20244.

them more or less suitable for the dynamic maintenance of a file. The use of K-dimensional binary trees [2] (or simply Kd-trees) is a convenient device for supporting associative queries because it usually supports a large set of operations with relatively simple algorithms and offers reasonable compromises on time and space requirements.

The expected-case analysis for Kd-trees is made under the assumption that the Kd-tree is built by successive insertions of records with K-dimensional keys independently drawn from a random source (say, uniformly distributed random K-tuples in $[0,1]^K$) [7]. We say that a Kd-tree built in this way is a *randomly built* Kd-tree. For randomly built Kd-trees of size n the expected time for an insertion, a deletion or an exact search of a record is $\mathcal{O}(\log n)$ [2]. Intersection queries and nearest neighbor queries are also supported by this data structure [2,3,4,13]. The efficient expected case performance of these operations holds only under the assumption that the Kd-tree is randomly built. Unfortunately, this assumption does not always hold: for instance, it fails when the keys to be inserted are sorted with respect to one of the attributes. Moreover, an alternation of deletions and insertions over a randomly built Kd-tree destroys randomness, in the sense that the resulting tree is no longer a randomly built tree. This happens even if every item of the file is equally likely to be deleted. After some updates the tree may need to be rebuilt to preserve the logarithmic expected performance.

One possibility to overcome this problem is the use of optimized Kd-trees [4,13], assuming that the file of records is given a priori. Such Kd-trees are perfectly balanced and their logarithmic performance in dictionary operations is guaranteed. However, when insertions and deletions are to be performed, a reorganization of the whole tree is required. Thus, this alternative is not suitable unless updates occur rarely and most records in the file are known in advance. Another approach is to introduce explicit constraints on the balancing of the trees, as in dynamically balanced Kd-trees and in divided Kd-trees [8,14]. Each update operation checks whether a predefined balance criterion remains true. If thresholds for balance criteria are exceeded, then a complex reorganization of the tree is performed. Unfortunately, these methods sacrifice the simplicity of the standard Kd-tree algorithms.

A third approach consists in the use of randomization to produce simple yet more robust algorithms. Randomization guarantees efficient expected performance that no longer depends on assumptions about the input distribution [10]. Randomization has been successfully applied to the design of dictionaries by Aragon and Seidel [1], Pugh [9] and Martínez and Roura [11].

In this paper we present randomized K-dimensional binary search trees (*randomized Kd-trees*, for short), a new type of K-dimensional binary trees that (1) support any sequence of update operations (insertions and deletions) while preserving the randomness of the tree, (2) do not demand preprocessing and (3) efficiently support exact match and associative queries. The main idea is to use randomization for performing insertions and deletions in regions of Kd-trees other than the leaves in such a way that the resulting tree is a randomly built tree. As a consequence, the expected case performance of every operation holds irrespective of the order of update operations since it only depends on the random choices made by the randomized update algorithms. A full version of this paper is accessible from: `http://www.lsi.upc.es/~techreps/1998.html`.

2 Relaxed Kd-trees

Multidimensional binary trees (or Kd-trees) [2] are a generalization of the well known binary search trees (BSTs), that handles records with keys of K attributes. In what follows and without loss of generality, we identify a record with its corresponding key as $\mathtt{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(K)})$, where each $x^{(i)}$, $1 \leq i \leq K$, refers to the value of the i-th attribute of the key $\mathtt{x}$. The domain $D = D_1 \times D_2 \times \ldots \times D_K$ of the keys is K-dimensional and each D_i is a totally ordered set.

Definition 1. *A* Kd-tree *for a set of keys is a binary tree in which:*
1.- Each node contains a K-dimensional key and has an associated discriminant $j \in \{1, 2, \ldots, K\}$.
2.- For every node with key $\mathtt{x}$ and discriminant j, any key $\mathtt{y}$ in the left subtree satisfies $y^{(j)} < x^{(j)}$ and any key $\mathtt{y}$ in the right subtree satisfies $y^{(j)} > x^{(j)}$.
3.- The root node has depth 0 and discriminant 1. All nodes at depth d have discriminant $(d \bmod K) + 1$.

Note that, if $K = 1$, then a Kd-tree is a binary search tree. As for binary search trees, it is assumed that each domain D_i is a continuous set and therefore the probability of having two equal values of the i-th attribute is zero [2,7]. Thus, any key $\mathtt{x}$ to be inserted is always *compatible* with a tree T in the sense that, for any i, its i-th attribute is different from the i-th attribute of any other key already in T.

The cyclic restriction of Kd-trees (Condition 3 in Definition 1) forces update operations to be very laborious or located at the leaves. But, as we shall show, in our quest for randomized Kd-trees, we add the flexibility of performing operations in places of Kd-trees other than the leaves without major reorganization.

With this purpose in mind, we now introduce relaxed Kd-trees. These are Kd-trees where Condition 3 in Definition 1 is dropped but where each node explicitly stores its discriminant. So, the sequence of discriminants in a path from the root to any leaf is arbitrary.

Definition 2. *A* relaxed Kd-tree *for a set of keys is a binary tree in which:*
1.- Each node contains a K-dimensional key and has associated a discriminant $j \in \{1, 2, \ldots, K\}$.
2.- For every node with key $\mathtt{x}$ and discriminant j, any key $\mathtt{y}$ in the right subtree satisfies $y^{(j)} < x^{(j)}$ and any key $\mathtt{y}$ in the left subtree satisfies $y^{(j)} > x^{(j)}$.

Again, if $K = 1$, a relaxed Kd-tree is a binary search tree. Now, we say that a relaxed Kd-tree of size n is *randomly built* if it is built by n insertions where the keys are drawn independently from a continuous distribution (for example, uniformly from $[0,1]^K$) and the discriminants are drawn uniformly and independently from $\{1,\ldots,K\}$. In randomly built relaxed Kd-trees, this assumption about the distribution of the input implies that the $n!K^n$ possible configurations of input file and discriminant sequence are equiprobable (observe that in randomly built Kd-trees the assumption implies that all $n!^K$ input sequences are equiprobable) [2,3,7].

In particular, in a randomly built relaxed Kd-tree each of the nK possibilities of (key,discriminant) pairs are equally likely to appear in the root and once the root is fixed, the left and right subtrees are independent randomly built relaxed Kd-trees. Thus, we obtain the same distribution for the shapes of randomly built relaxed Kd-trees as for randomly built BSTs and randomly built Kd-trees. Moreover, given that j keys are in the left subtree, the distribution of randomly built relaxed Kd-trees of size n has the recursive property that the distribution of left and right subtrees is the distribution of randomly built relaxed Kd-trees with sizes j and $n-1-j$, for all $0 \le j < n$, and the probability that the left subtree has size j is simply $1/n$, independently of K.

Parameters like the internal or the external path length do not depend on the discriminants [6,7] and hence their expected value is the same for randomly built BSTs and randomly built relaxed Kd-trees.

Theorem 1. [6] *The expected internal and external path length of a randomly built relaxed Kd-tree is $\mathcal{O}(n\log n)$.*

Therefore, randomly built relaxed Kd-trees perform like randomly built Kd-trees for exact match queries. Such an operation explores a path, that, because of Theorem 1 is logarithmic in the expected case.

The algorithms for associative queries in relaxed Kd-trees are similar to those for Kd-trees but their expected-case performances are not necessarily the same (partial match, for instance), since, for randomly built relaxed Kd-trees a path down the tree does not have a cyclic pattern of discriminants but a random sequence of discriminants. We now analyze the average cost of performing partial match and nearest neighbor queries in relaxed Kd-trees.

A *partial match query* is a query in which only s out of the K attributes of a given key are specified (with $0 < s < K$). The retrieval consists of finding all the records whose specified key attributes coincide with those of the given query.

A partial match query in a relaxed Kd-tree explores the tree in the following way. At each node it verifies the corresponding discriminant. If it is specified (which happens with probability s/K), then it follows recursively only one of the subtrees depending on its attribute. Otherwise (with probability $1-s/K$), it follows the two subtrees recursively. Hence we have the following theorem.

Theorem 2. *The expected cost P_n, measured as the number of comparisons, of a partial match query with s out of K attributes specified in a randomly built relaxed Kd-tree with n nodes, is $P_n = \beta n^{\delta} + \mathcal{O}(1)$, where*

$$\delta = \delta(s/K) = -\frac{1}{2} + \frac{\Delta(s/K)}{2} \sim 1 - \frac{2s}{3K} + \mathcal{O}\left((s/K)^2\right),$$
$$\beta = \frac{\Gamma(2\delta+1)}{(1-s/K)(\delta+1)\Gamma^3(\delta+1)},$$

with $\Delta(x) = \sqrt{9-8x}$ and $\Gamma(x)$ the Gamma function [5].

The proof of this theorem appears in the full version of this paper. The cost is obtained by solving a recurrence by means of ordinary generating functions and complex singularity analysis.

The expected cost of partial match queries in randomly built relaxed Kd-trees is slightly higher than the one given by Flajolet and Puech [3] for randomly built Kd-trees. Notice also that the constant β in the main order term of the expected cost of partial matches in relaxed Kd-trees is independent of the pattern of specified/unspecified attributes, whereas for standard Kd-trees such a constant is dependent on the particular pattern of the query [3].

A request for the closest record in a file to a given query record, under a determined distance function, is called a *nearest neighbor* query. More generally, given a multidimensional record $\mathtt{q}$, a distance d over D, and a relaxed Kd-tree T of n records, an m-nearest neighbor query consists of finding the m points $\mathtt{y}$ in T such that $d(\mathtt{q},\mathtt{y}) \leq d(\mathtt{q},\mathtt{x})$, for all $\mathtt{x}$ but $m-1$ of the records in T. The nearest neighbor query is the case $m=1$.

Theorem 3. *The expected cost (measured as the number of visited nodes) of a nearest neighbor query in a randomly built relaxed Kd-tree with n nodes, is $\mathcal{O}(\log n)$ time.*

Sketch of the Proof. The proof is similar to the proof given by Friedman, Bentley and Finkel [4]; it is only required to observe that the expected volume of a cell in the partition induced by randomly built relaxed Kd-trees is $1/(n+1)$.□

3 Randomized Kd-trees

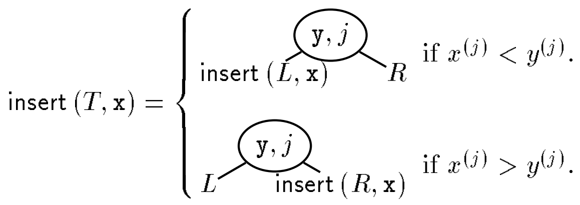
Relaxed Kd-trees have been shown to efficiently support dictionary operations and associative queries in the expected-case. However, we still have the assumption that the input sequences of keys are independently drawn from any continuous probability distribution. In what follows we drop this requirement via randomized algorithms. The randomized algorithms presented here require that each node stores the size of the subtree beneath it, because the behavior of our randomized operations depends on the size of the subtrees to which they are applied.

We say that a relaxed Kd-tree is a *randomized Kd-tree* if it is the result of a sequence of update operations performed by means of the randomized algorithms introduced below, applied to an initially empty tree. We shall show in this section that any tree obtained this way is a randomly built relaxed Kd-tree.

Informally, in order to produce a randomly built relaxed Kd-tree two properties must hold: (1) a new inserted key should have some probability of becoming the root of the tree, or the root of one of the subtrees of the root, and so forth and (2) every discriminant should have the same probability of being the discriminant of the new node.

The insertion of $\mathtt{x}$ in a relaxed Kd-tree T, begins by generating uniformly a random integer $i \in \{1, \dots, k\}$. This step corresponds to the assignment of a discriminant to the new node. If the tree T is the empty tree, then the algorithm insert produces a tree with root node $(\mathtt{x}, i)$ and empty left and right subtrees (we say that a node contains $(\mathtt{x}, i)$ if its key is $\mathtt{x}$ and its discriminant is i).

If the tree T is not empty, then, with probability $\frac{1}{n+1}$ the (key, discriminant) pair $(\mathtt{x}, i)$ must be placed at the root of the new tree using the insert_at_root algorithm (since the new tree will have size $n+1$). Otherwise, we insert the pair $(\mathtt{x}, i)$ recursively in either the left or the right subtree of T depending on $\mathtt{x}$'s order relation with the root of T. Let T have root $(\mathtt{y}, j)$, left subtree L and right subtree R. Symbolically we have, with probability $\frac{1}{n+1}$, insert $(T, \mathtt{x}) =$ insert_at_root $(T, \mathtt{x}, i)$, and, with probability $\frac{n}{n+1}$,



The algorithm insert requires the possibility of inserting the pair $(\mathtt{x}, i)$ at the root of any subtree of a relaxed Kd-tree T (using insert_at_root). If T is empty, then, insert_at_root $(T, \mathtt{x}, i)$ gives as a result a tree with root node $\mathtt{x}$, discriminant i and empty left and right subtrees. When T is not empty, by definition, insert_at_root $(T, \mathtt{x}, i) = T'$ where the root of T' is $(\mathtt{x}, i)$, its left subtree consists of all the elements of T with i-th dimensional key smaller than $x^{(i)}$ and its right subtree contains those elements of T with i-th dimensional key greater than $x^{(i)}$. To obtain the left and right subtrees of T' we use the split algorithm, which we present later.

The deletion of a record from a randomly built relaxed Kd-tree consists of searching in the tree the key $\mathtt{x}$ to be deleted and then joining its corresponding left and right subtrees. Informally, in order to keep a randomly built relaxed Kd-tree all the nodes in these subtrees should have some probability of taking the place of the deleted node. This is achieved by the join algorithm (which we introduce later). Let $\mathtt{x}$ be the record to be deleted from the randomly built relaxed Kd-tree T. Let T have root $(\mathtt{y}, j)$, left subtree L and right subtree R. Symbolically,

$$\mathsf{delete}\,(T,\mathtt{x}) = \begin{cases} \begin{tikzcd}[column sep=small] & (\mathtt{y}, j) \arrow[dl, dash] \arrow[dr, dash] & \\ \mathsf{delete}\,(L,\mathtt{x}) & & R \end{tikzcd} & \text{if } \mathtt{x} \neq \mathtt{y} \text{ and } x^{(j)} < y^{(j)}. \\ \begin{tikzcd}[column sep=small] & (\mathtt{y}, j) \arrow[dl, dash] \arrow[dr, dash] & \\ L & & \mathsf{delete}\,(R,\mathtt{x}) \end{tikzcd} & \text{if } \mathtt{x} \neq \mathtt{y} \text{ and } x^{(j)} > y^{(j)}. \\ \mathsf{join}\,(L,R,j) & \text{if } \mathtt{x} = \mathtt{y}. \end{cases}$$

Both, insertions and deletions, consist of two different steps. A first step in which one must follow a path in the tree in order to locate the place where the key must be inserted or deleted and, a second step in which the update is performed.

The expected cost of insertions and deletions is $\mathcal{O}(\log n)$ time, since both the insertion at the root of any element and the deletion of any element occur near the leaves on the average. Thus, the reconstruction step, although expensive, occurs in subtrees of expected size $\mathcal{O}(\log n)$ [2].

The split algorithm receives as parameters a tree T, a key $\mathtt{x}$ and a discriminant i, and produces two different trees: $T_{<_i} = \mathsf{split}_<\,(T,\mathtt{x},i)$ containing all the elements of T with i-th attribute smaller than $x^{(i)}$, and $T_{>_i} = \mathsf{split}_>\,(T,\mathtt{x},i)$ containing all the elements of T with i-th attribute greater than $x^{(i)}$.

The algorithm $\mathsf{split}_<$ works in the following way: When T is the empty tree, the $\mathsf{split}_<$ algorithm returns the empty tree. Otherwise, let T have root $(\mathtt{y}, j)$, left subtree L, and right subtree R. We have four cases to consider: (1) If $i = j$ and $y^{(i)} < x^{(i)}$, then $\mathtt{y}$ belongs to $T_{<_i}$, all the elements of L do as well (since each i-th dimensional key in L is smaller than $y^{(i)}$ and thus smaller than $x^{(i)}$) and the operation proceeds recursively in R to complete the result; (2) If $i = j$ and $y^{(i)} > x^{(i)}$, then the algorithm proceeds recursively in L (since $\mathtt{y}$ and all the elements in R are greater than $x^{(i)}$); (3) If $i \neq j$ and $y^{(i)} < x^{(i)}$, then $\mathtt{y}$ belongs to $T_{<_i}$ and the algorithm proceeds recursively in L and in R and attach to $\mathtt{y}$ the results of splitting L and R; (4) If $i \neq j$ and $y^{(i)} > x^{(i)}$, then $\mathtt{y}$ is not in the tree and the algorithm must proceed recursively in L and R, but now it has to join the trees resulting from splitting L and R. Symbolically, if T is not empty,

$$\mathsf{split}_<\,(T,\mathtt{x},i) = \begin{cases} \begin{tikzcd}[column sep=small] & (\mathtt{y}, j) \arrow[dl, dash] \arrow[dr, dash] & \\ L & & \mathsf{split}_<\,(R,\mathtt{x},i) \end{tikzcd} & \text{if } i = j \text{ and } y^{(i)} < x^{(i)}, \\ \mathsf{split}_<\,(L,\mathtt{x},i) & \text{if } i = j \text{ and } y^{(i)} > x^{(i)}, \\ \begin{tikzcd}[column sep=small] & (\mathtt{y}, j) \arrow[dl, dash] \arrow[dr, dash] & \\ \mathsf{split}_<\,(L,\mathtt{x},i) & & \mathsf{split}_<\,(R,\mathtt{x},i) \end{tikzcd} & \text{if } i \neq j \text{ and } y^{(i)} < x^{(i)}, \\ \mathsf{join}\,(\mathsf{split}_<\,(L,\mathtt{x},i)\,, \mathsf{split}_<\,(R,\mathtt{x},i)\,, j) & \text{if } i \neq j \text{ and } y^{(i)} > x^{(i)}. \end{cases}$$

The $\mathsf{split}_>$ operation is defined symmetrically. It is not difficult to see that both $\mathsf{split}_<\,(T,\mathtt{x},i)$ and $\mathsf{split}_>\,(T,\mathtt{x},i)$ compare $\mathtt{x}$ against the same keys in T as if we were performing a partial match with one attribute specified. But the additional cost of the join algorithm must be taken into account.

We now describe the algorithm join (A, B, j). By definition this algorithm is applied only when the j-th attribute values of the tree A are smaller than the j-th attribute values of the tree B. Informally, in order to produce a randomly built relaxed Kd-tree, each node of A and each node of B must have some probability of becoming the root of the new tree. If A and B have sizes n and m respectively, then, $T = \text{join}\,(A, B, j)$ has size $n + m$. Thus, the join algorithm selects with probability $\frac{n}{n+m}$ the root of A, (a, i_a), as root of the resultant tree T and with complementary probability the root,(b, i_b), of B. We have the following three cases to consider: (1) If A and B are both empty, then, $T = \text{join}\,(A, B, j)$ is the empty tree; (2) If only one of them is empty, then T is equal to the non-empty one; (3) If A and B are both non-empty trees with roots (a, i_a), (b, i_b), left subtrees L_a, L_b and right subtrees R_a and R_b, respectively, then we face four subcases. Two subcases correspond to the situation where (a, i_a) has been chosen as the root of T: (3a) If $j = i_a$, then the left subtree of T is L_a and its right subtree is the result of joining R_a with B (since every j-th dimensional key in B is greater than $a^{(i_a)}$); (3b) If $j \neq i_a$, then the left subtree of T is the result of joining L_a with $\text{split}_<\,(B, \mathsf{a}, i_a)$. Symmetrically, the right subtree of T is the result of joining R_a with $\text{split}_>\,(B, \mathsf{a}, i_a)$. The two complementary subcases occur when (b, i_b) is selected as the root of T. Symbolically, with probability $\frac{n}{n+m}$, we have,

$$\text{join}\,(A, B, j) = \begin{cases} \begin{tikzcd}[column sep=small] & \boxed{\mathsf{a}, i_a} \arrow[dl, dash] \arrow[dr, dash] & \\ L_a & & \text{join}\,(R_a, B, j) \end{tikzcd} & \text{if } j = i_a, \\ \begin{tikzcd}[column sep=small] & \boxed{\mathsf{a}, i_a} \arrow[dl, dash] \arrow[dr, dash] & \\ \text{join}\,(L_a, \text{split}_<\,(B, \mathsf{a}, i_a), j) & & \text{join}\,(R_a, \text{split}_>\,(B, \mathsf{a}, i_a), j) \end{tikzcd} & \text{if } j \neq i_a. \end{cases}$$

And with probability $\frac{m}{n+m}$, when (b, i_b) is selected as the root of T, we obtain symmetrical equations.

A join traverses the tree in a similar way than the partial match algorithm with one specified attribute, with the additional cost of the split algorithm.

The randomized split and join algorithms preserve the randomness of their input (because of Lemma 1 below). In other words, when applied to randomly built relaxed Kd-trees, both the split and the join algorithms produce randomly built relaxed Kd-trees. Moreover, since this happens, the insert and delete algorithms when applied to randomly built relaxed Kd-trees produce randomly built relaxed Kd-trees. These claims are made explicit in Theorems 4 and 5 below.

Lemma 1. *Let T be a relaxed Kd-tree with root $(\mathtt{y}, j)$ and let $T_{<_i}$ and $T_{>_i}$ be the relaxed Kd-trees produced by* $\text{split}_<\,(T, \mathtt{x}, i)$ *and* $\text{split}_>\,(T, \mathtt{x}, i)$*, respectively, where $\mathtt{x}$ is any key compatible with T. Let T' be the relaxed Kd-tree produced by* join (A, B, j)*, where A and B are subtrees of T. If T is a randomly built relaxed Kd-tree, then $T_{<_i}$ and $T_{>_i}$ are independent randomly built relaxed Kd-trees. Also, if T is randomly built then T' is a randomly built relaxed Kd-tree.*

Sketch of the Proof. The proof is by induction on the size of the tree and can be found in the full version of this paper.□

Theorem 4. *If T is a randomly built relaxed Kd-tree that contains the set of keys X and* $\mathtt{x}$ *is any key compatible with X, then* $\mathsf{insert}(T, \mathtt{x})$ *returns the randomly built relaxed Kd-tree containing the set of keys $X \cup \{\mathtt{x}\}$.*

Proof. The proof is by induction on the size n of T. If $n = 0$, then T is the empty tree (a randomly built relaxed Kd-tree), and $\mathsf{insert}(T, \mathtt{x})$ returns a randomly built relaxed Kd-tree with root $(\mathtt{x}, i)$, and two empty subtrees, where i is uniformly generated from $\{1, \ldots, K\}$. We assume now that T is not empty and that the theorem is true for all sizes $< n$. Consider an item $(\mathtt{y}, j) \in T$. The probability that $(\mathtt{y}, j)$ is the root of T before the insertion of $\mathtt{x}$ is $\frac{1}{Kn}$, since T is a randomly built Kd-tree. The probability that $(\mathtt{y}, j)$ is at the root after the insertion of $\mathtt{x}$ is the probability that $\mathtt{x}$ is not at the root and $(\mathtt{y}, j)$ was at the root, which is $\frac{1}{Kn} \times \frac{n}{n+1}$, resulting in the desired probability. Moreover, if $(\mathtt{x}, i)$ is not inserted at the root at the first step, the insertion proceeds recursively in either the left or the right subtrees of T. By the inductive hypothesis, this result in a randomly built relaxed Kd-tree. Finally, with probability $\frac{1}{n+1}$, $(\mathtt{x}, i)$ is inserted at the root of T. Since i is the discriminant of $\mathtt{x}$ with probability $\frac{1}{K}$, the resulting probability is $\frac{1}{K(n+1)}$ as expected, and because of Lemma 1 the two subtrees attached to the new root are randomly built relaxed Kd-trees.□

Theorem 5. *If T is a randomly built relaxed Kd-tree that contains the set of keys X, then,* $\mathsf{delete}(T, \mathtt{x})$ *produces a randomly built relaxed Kd-tree T' that contains the set of keys $X \backslash \{\mathtt{x}\}$.*

Proof. If the key $\mathtt{x}$ is not in T, then the algorithm does not modify the tree, which is randomly built. Let us now suppose that $\mathtt{x}$ is in T, this case is proved by induction on the size of the tree. If $n = 1$, $\mathtt{x}$ is the only key in the tree and after deletion we obtain the empty tree which is a randomly built relaxed Kd-tree. We assume that $n > 1$ and that the theorem is true for all sizes $< n$. If $\mathtt{x}$ was not the key at the root of T we proceed recursively either in the left or in the right subtrees, and by inductive hypothesis we obtain a randomly built subtree. If $\mathtt{x}$ was the key at the root of T, the tree after deletion is the result of joining the left and right subtrees of T, which produce a randomly built relaxed Kd-tree because of Lemma 1. Finally, after deletion, each node has probability $\frac{1}{K(n-1)}$ of being the root. Observe that this is the desired probability since the new tree has size $(n - 1)$.□
Combining these two theorems we obtain the following important corollary.

Corollary 1. *The result of any arbitrary sequence of insertions and deletions, starting from an initially empty tree is always a randomly built relaxed Kd-tree.*

It is important to observe that using these new update operations we have that the expected running time for insertions, deletions, partial match and nearest neighbor queries depends only on the random choices made by the randomized update algorithms and not on the sequence of updates.

4 Final Remarks

Throughout this work we have presented a randomized generalization of Kd-trees (the randomized Kd-trees) that inherits the simplicity, robustness, flexibility and efficiency of standard Kd-trees, yet essentially preserves (in terms of the number n of records) the expected performance of all the operations (updates and associative queries) of Kd-trees independently of the order in which updates (insertions and deletions) are performed. None of the update operations or queries require preprocessing. Finally, randomized Kd-trees are also efficient in space (it is $\mathcal{O}(n)$), requiring slightly more space than standard Kd-trees, as the randomized Kd-trees need to store, for each node, its discriminant and the size of the subtree beneath it.

Acknowledgments

We thank Carme Àlvarez, Salvador Roura, Fatos Xhafa and anonymous referees for their useful comments on earlier versions of this work. We also thank H. Prodinger for technical advice and comments about the analysis of partial match queries.

References

1. Aragon, C. R., Seidel, R. G.: Randomized Search Trees. Algorithmica **16** (1996) 464–497 200
2. Bentley, J. L.: Multidimensional binary search trees used for associative retrieval. Comm. of the ACM **18(9)** (1975) 509–517 200, 201, 202, 205
3. Flajolet, Ph., Puech, C.: Partial match retrieval of multidimensional data. J. of the ACM **33(2)** (1986) 371–407 200, 202, 203
4. Friedman, J. H., Bentley, J. L., Finkel, R. A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. on Mathematical Software **3(3)** (1977) 209–226 200, 203
5. Graham, R. L., Knuth, D. E., Patashnik, O.: Concrete Mathematics. Addison-Wesley (1989) 203
6. Knuth, D. E.: The Art of Computer Programming, vol. 3: Sorting and Searching, 2nd. edition. Addison-Wesley (1998) 199, 202
7. Mahmoud, H. M.: Evolution of Random Search Trees. Wiley (1992) 200, 201, 202
8. Overmars, M. H., van Leewen, J.: Dynamic multi-dimensional data structures based on quad- and k-d-trees. Acta Informatica **17(3)** (1982) 267–285 200
9. Pugh, W.: Skip list: A probabilistic alternative to balanced trees. Comm. of the ACM **33(6)** (1990) 668–676 200
10. Raghavan, P., Motwani, R.: Randomized Algorithms. Cambridge Univ. Press (1995) 200
11. Roura, S., Martínez, C.: Randomized binary search trees. J. of the ACM **45(2)** (1998) 288–323 200
12. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley (1990) 199

13. Sproull, R.: Refinements to nearest neighbor searching in k-dimensional trees. Algorithmica **6** (1991) 579–589 200
14. van Kreveld, M. J., Overmars, M. H.: Divided k-d-trees. Algorithmica **6** (1991) 840–858 200

Randomized O(log log n)-Round Leader Election Protocols in Packet Radio Networks*

Koji Nakano[1] and Stephan Olariu[2]

[1] Department of Electrical and Computer Engineering
Nagoya Institute of Technology
Showa-ku, Nagoya 466-8555, Japan
[2] Department of Computer Science, Old Dominion University
Norfolk, VA 23529, USA

Abstract. The main contribution of this work is to propose efficient randomized leader election protocols in Packet Radio Networks (PRN). We show that in a one-channel PRN a leader can be elected among n identical stations in $O(\log\log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{\log n})$ or in $O(\log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{n})$. We begin by designing such a protocol for the case where the number n of stations is known beforehand. Next, we show that the leader election can be completed within the same number of broadcast rounds even if n is not known. We show that our protocols are optimal in the sense that no randomized protocol that terminates in $o(\log n)$ rounds can elect a leader with probability higher than $1 - O(\frac{1}{n})$ and no randomized protocol that terminates in $o(\log\log n)$ rounds can elect a leader with probability higher than $1 - O(\frac{1}{\log n})$.

1 Introduction

Let $\mathcal{S}$ be a set of n radio transceivers in the Packet Radio Network (PRN), henceforth referred to as *stations*. The stations are identical and cannot be distinguished by serial or manufacturing number. The problem that we address in this work is the well-known *leader election* problem which asks to designate one of the station as *leader* [2,9]. In other words, after executing the leader election protocol, exactly one station learns that it was elected as leader, while the remaining stations learn the identity of the leader elected.

As customary, the time is assumed to be slotted and all the stations have a local clock that keeps (synchronous) time [3]. Also, all broadcast operations are performed at time slot boundaries. The stations are assumed to have the computing power of a usual laptop computer; in particular, they all run the same protocol and can generate random bits that provide "local data" on which the stations may perform computations.

The stations communicate using k communication channels denoted by C(1), C(2), ..., C(k). At any one time, a station can tune in to one such channel and/or

* Work supported in part by the National Science Foundation under grant CCR-9522093, and by ONR grant N00014-97-1-0526.

broadcast on at most one (possibly the same) channel. A broadcast operation involves a data packet whose length is such that the broadcast operation can be completed within one time slot. In any given time slot the *status* of a radio channel is:

– *NULL* if no station broadcasts on the channel in the current time slot
– *SINGLE* if exactly one station broadcasts on the channel in the current time slot, and
– *COLLISION* if two or more stations broadcast on the channel in the current time slot.

Suppose that the PRN has n stations and that the i-th station, $(1 \leq i \leq n)$, participates in the protocol for r_i broadcast rounds. Note that the i-th station need not broadcast data in all of these r_i rounds. The performance of the protocol is evaluated by the following two metrics:

– *broadcast complexity* r defined as the maximum number of broadcast rounds in which any station participates. In other words, $r = \max\{r_1, r_2, \ldots, r_n\}$;
– *work complexity* w of the protocol defined as $w = r_1 + r_2 + \cdots + r_n$.

Clearly, if all the stations participate in the protocol for the same number of broadcast rounds we have $r = r_1 = r_2 = \cdots = r_n$ and $w = rn$. However, in case most of the stations terminate earlier, $w \ll rn$ and these stations are available to participate in other protocols.

Given its importance, the problem of designing efficient protocols for PRN has been addressed in the literature [2,3,4,5]. However, some of the existing protocols are designed under the assumption that the n stations in the PRN have been assigned *consecutive* numbers, typically, from 1 to n. The highly non-trivial task of numbering the stations in this fashion is often ignored in the literature. The *leader election* problem is even more fundamental since the consecutive numbering problem we just mentioned can be solved by repeatedly selecting a leader in a subset of the stations.

It is customary to address the leader election problem in three different scenarios:

Scenario 1: The number n of stations in the PRN is known in advance;
Scenario 2: The number n of stations in the PRN is unknown but an upper bound u on this number is known in advance;
Scenario 3: Neither the number of stations nor an upper bound on this number is known in advance.

It is intuitively clear that the task of leader election is the easiest in Scenario 1 and the hardest in Scenario 3, with Scenario 2 being in-between the two. In this work we shall not deal with broadcast protocols satisfying the conditions of Scenario 2.

Several randomized protocols using a one-channel PRN have been presented in the literature. Metcalfe and Boggs [7] presented a leader election protocol for Scenario 1 that is guaranteed to terminate in $O(1)$ expected rounds. Later, Willard [9] showed that the leader election can be solved in $\log\log u + O(1)$ expected rounds if the conditions of Scenario 2 hold. Willard's protocol involves two stages: the first stage, using binary search, guesses in $\log\log u$ rounds a number i, $(0 \leq i \leq \log u)$, satisfying $2^i \leq n < 2^{i+1}$. The second stage finds the leader in $O(1)$ expected rounds using the protocol of [7]. Thus, the protocol finds the leader in $\log\log u + O(1)$ expected rounds. Willard [9] also improved this protocol to run under the conditions of Scenario 3 in $\log\log n + o(\log\log n)$ expected rounds. The first stage of the improved protocol uses the technique presented by Bentley and Yao [1], which finds an integer i satisfying $2^i \leq n < 2^{i+1}$ for unknown n. However, all the stations executing this protocol terminate at the same time, implying that the work complexity of the protocol is $O(n \log\log n)$.

The main contribution of this work is to present randomized protocols to solve the leader election problem under the conditions of Scenario 3 on a one-channel PRN. We begin by showing that, in case Scenario 1 applies, the $O(1)$ expected round protocol of [7] can elect a leader in $O(\log\log n)$ broadcast rounds with probability at least $1-O(\frac{1}{\log n})$ or in $O(\log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{n})$. Further, we show that this protocol is optimal in the sense that no $o(\log n)$-round (resp. $o(\log\log n)$-time) randomized election protocol can elect a leader with probability $1 - O(\frac{1}{\log n})$ (resp. $1 - O(\frac{1}{\log n})$).

We then go on to show that the leader election problem can be solved within the same bounds even if the conditions of Scenario 3 are satisfied, that is, if neither the number n of stations nor an upper bound on this number is known in advance. Specifically, we show that the leader election problem can be solved in $O(\log\log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{\log n})$ or in $O(\log n)$ broadcast rounds with probability at least $1-O(\frac{1}{n})$, even if no station knows n at the beginning of the protocol. Our protocols for the case of Scenario 3 are also optimal, because the lower bound discussed above holds for this case as well. We further show that our leader election protocols have an $O(n)$ work complexity.

We also provide simulation results to show that our randomized $O(\log\log n)$-round leader election protocol performs extremely well in practice. In fact, our simulation results show that our protocol features a very small proportionality factor, being eminently of practical relevance.

To put our work in perspective, we note that our leader election protocols improve on the result of Willard [9] in three respects: First, our protocol terminate in $O(\log\log n)$ broadcast rounds with *high probability*, whereas the one in [9] terminates in $O(\log\log n)$ expected number of rounds. Second, our protocol has a work complexity of $O(n)$, as opposed to that in [9] that requires $O(n \log\log n)$ work. Third, our protocol is more uniform, more intuitive, easier to understand and to reason about.

2 Leader Election for Known n

This section reviews basic probability theory results that are useful for analyzing the performance of our randomized leader election protocols. For a more detailed discussion of background material we refer the reader to [6,8]. We then review the simple leader election protocol of [7] for known n and show that this simple protocol that terminates in $O(1)$ expected broadcast rounds can be also seen to terminate in $O(\log n)$ or $O(\log\log n)$ rounds with high probability. Finally, we prove the optimality of this protocol by presenting a tight lower bound.

Throughout, $\Pr[A]$ will denote the probability of event A. For a random variable X, $E[X]$ denotes the expected value of X. Let X be a random variable denoting the number of successes in n independent Bernoulli trials with parameters p and $1-p$. It is well known that X has a *binomial distribution* and that for every r, $(0 \leq r \leq n)$,

$$\Pr[X = r] \;=\; \binom{n}{r} p^r (1-p)^{n-r}. \tag{1}$$

Further, the expected value of X is given by

$$E[X] = \sum_{r=0}^{n} r \cdot \Pr[X = r] = np. \tag{2}$$

To analyze the tail of the binomial distribution, we shall make use of the following estimates, commonly referred to as *Chernoff bounds*:

$$\Pr[X \leq (1-\epsilon)E[X]] \leq e^{-\frac{\epsilon^2}{2}E[X]} \qquad (0 \leq \epsilon \leq 1). \tag{3}$$

$$\Pr[X \geq (1+\epsilon)E[X]] \leq e^{-\frac{\epsilon^2}{3}E[X]} \qquad (0 \leq \epsilon \leq 1). \tag{4}$$

We now review the simple randomized election protocol of [7] for the case of Scenario 1, that is, when the number n of stations is known in advance. The details follow.

```
Protocol Election-for-known-n
repeat
    each station broadcasts on channel C(1) with probability $\frac{1}{n}$;
until the status of channel C(1) is SINGLE;
the station that has broadcast in the last round is declared the leader.
```

The correctness of `Election-for-known-n` is easily seen. The protocol continues until, at some point, only one station broadcasts on the channel. Clearly, upon this broadcast, the leader is known by all the participating stations.

Thus, we now turn to the problem of estimating the number of broadcast rounds necessary by the protocol to terminate. Let X be the random variable denoting the number of stations broadcasting in a given round. Then, by virtue of (1), at the end of this round the status of the channel is SINGLE with probability

$$\Pr[X = 1] = \binom{n}{1}\left(\frac{1}{n}\right)^1\left(1-\frac{1}{n}\right)^{n-1} = \left(1-\frac{1}{n}\right)^{n-1} > \frac{1}{e}.$$

To put it differently, any given broadcast round fails to find a leader with probability at most $1 - \frac{1}{e}$.

This first result allows us to observe that the expected number of broadcast rounds required by protocol `Election-for-known-n` to terminate is bounded by

$$\sum_{t=1}^{\infty} t \cdot \left(1 - \frac{1}{e}\right)^{t-1} \left(\frac{1}{e}\right) = e.$$

Thus, `Election-for-known-n` elects a leader in e expected rounds.

Now, consider the first t rounds in the protocol `Election-for-known-n`. The probability that none of the first t rounds produces a leader is

$$(\Pr[X \neq 1])^t < \left(1 - \frac{1}{e}\right)^t \leq e^{-\frac{t}{e}}. \qquad \text{(since } 1 + x \leq e^x\text{)}$$

Consequently, if we take[1] $t = e \ln n$, the probability that `Election-for-known-n` will fail to elect a leader at the end of $O(\log n)$ broadcast rounds is at most $\frac{1}{n}$. Similarly, if we take $t = e \ln \log n$, the probability that the protocol fails to identify a leader at the end of $O(\log \log n)$ broadcast rounds is at most $\frac{1}{\log n}$. Thus, we have proved the following result.

Lemma 1. *The protocol* `Election-for-known-n` *returns a leader in e expected broadcast rounds. It also returns a leader in $O(\log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{n})$ or in $O(\log \log n)$ broadcast rounds with probability at least $1 - O(\frac{1}{\log n})$.*

Consider a one-channel PRN with only two stations. Consider an arbitrary leader election protocol $\mathcal{P}$ and let us evaluate the number of broadcast rounds needed by $\mathcal{P}$ to terminate, that is, to break the symmetry inherent in the problem of leader election. In the first round, since the two stations are identical, each of them decides whether it should broadcast with the same probability p. Then, to break the symmetry, one of them should broadcast while the other should not. The probability of this event occurring is $2p(1-p) \leq \frac{1}{2}$. Thus, a leader is elected in one round with probability at most $\frac{1}{2}$. If the first round does not lead to the election of a leader the stations proceed to the second and, possibly, subsequent rounds. Since in any broadcast the data received from the channel is the same for the two stations, their broadcasting probability in any round must always be the same. Thus, in any given round, a leader is elected with probability at most $\frac{1}{2}$. It follows that, with probability at least $(\frac{1}{2})^t$, the first t rounds fail to elect a leader. Therefore, if the protocol $\mathcal{P}$ terminates in $o(\log n)$ rounds it must must fail to find a leader with probability at least $(\frac{1}{2})^{o(\log n)} \geq \omega(\frac{1}{n})$. Similarly, if $\mathcal{P}$ terminates in $o(\log \log n)$ rounds it must fail to find a leader with probability at least $\omega(\frac{1}{\log n})$. Thus, we have proved the following important result.

Theorem 1. *No leader election protocol that terminates in $o(\log n)$ rounds correctly returns the leader with probability higher than $1 - \frac{1}{n}$. Likewise, no leader*

[1] In this work log and ln are used to denote the logarithms to base 2 and e, respectively.

election protocol that terminates in $o(\log\log n)$ *rounds correctly returns the leader with probability higher than* $1 - \frac{1}{\log n}$.

3 Leader Election for Unknown n

The main goal of this section is to show that the performance of the leader election protocol devised in the Section 2 can be attained even if the conditions of Scenario 3 hold, that is, even if neither the number n of stations participating in the protocol, nor or an upper bound on this number, is known in advance.

3.1 Electing a Leader in Logarithmic Time

In this subsection we discuss an $O(\log n)$-round leader election protocol in the case where the number n of stations is not known in advance. Initially, all the stations participate in the protocol. In each round some of the stations quit and, thus, stop participating in the protocol. This continues until only one station remains that will become the leader.

```
Protocol Logarithmic-leader-election
repeat
    each station broadcasts on channel C(1) with probability $\frac{1}{2}$;
    if the status of channel C(1) is COLLISION or SINGLE then
        the stations that did not broadcast quit the protocol;
    endif
until the status of channel C(1) is SINGLE;
the station that has broadcast in the last round is declared the leader.
```

The correctness of protocol `Logarithmic-leader-election` being easily seen, we now turn to the task of evaluating the number of broadcast rounds it takes the protocol to terminate. Let N be the number of *remaining stations*, that is the stations still participating in the protocol at the beginning of a given broadcast round. Since the status of the channel C(1) in the previous round must have been COLLISION, it is the case that $N \geq 2$.

Let X be the random variable denoting the number of remaining stations at the end of the current round. We say that the current round is *successful* if $1 \leq X \leq \lceil \frac{N}{2} \rceil$. In this terminology, the probability that the current round is successful is

$$\Pr[1 \leq X \leq \lceil \frac{N}{2} \rceil] \geq \frac{1}{2} - \Pr[X = 0] = \frac{1}{2} - \frac{1}{2^N} \geq \frac{1}{4} \qquad \text{(since } N \geq 2\text{)}.$$

Since a successful round decreases the number of stations by at least half, $\lceil \log n \rceil$ successful rounds are sufficient to elect a leader. Our plan is to prove that, with probability at least $1 - \frac{1}{n}$, there are at least $\lceil \log n \rceil$ successful rounds among the first $16\lceil \log n \rceil$ rounds. For this purpose, let Y be the random variable denoting the number of successful rounds among $16\lceil \log n \rceil$ rounds. It is important to note that

$$E[Y] \geq 16\lceil \log n \rceil \cdot \frac{1}{4} \geq 4\lceil \log n \rceil.$$

Now, the Chernoff bound in (3) allows us to write for $0 \leq \epsilon \leq 1$

$$\Pr[Y < 4(1-\epsilon)\lceil \log n \rceil] \leq \Pr[Y < (1-\epsilon)E[Y]] \leq e^{-\frac{\epsilon^2}{2}E[Y]} \leq e^{-2\epsilon^2 \lceil \log n \rceil}.$$

In particular, for $\epsilon = \frac{3}{4}$ we have, $\Pr[Y < \lceil \log n \rceil] \leq e^{-\frac{9}{8}\lceil \log n \rceil} < e^{-\lceil \ln n \rceil} \leq \frac{1}{n}$. Consequently, we have proved that with probability at least $1 - \frac{1}{n}$, the first $16\lceil \log n \rceil$ rounds of the protocol `Logarithmic-leader-election` contain at least $\lceil \log n \rceil$ successful rounds. In turn, this shows that with probability at least $1 - \frac{1}{n}$ the protocol will terminate in the first $16\lceil \log n \rceil$ rounds. To summarize, we have proved the following result.

Theorem 2. *With probability at least* $1 - \frac{1}{n}$*, the protocol* `Logarithmic-leader-election` *elects a leader in* $16\lceil \log n \rceil$ *rounds.*

Our next goal is to show that the work complexity of `Logarithmic-leader-election` is bounded by $O(n)$. For this purpose, let N be the number of stations remaining at the beginning of the current round, and let X be the random variable denoting the number of stations that broadcast in this round. Clearly, $E[X] = \frac{N}{2}$; now using the bound in (4) with $\epsilon = \frac{1}{2}$, we have, $\Pr[X \geq \frac{3N}{4}] \leq e^{-\frac{N}{24}} < O\left(\frac{1}{N^2}\right)$. Further, $\Pr[X = 0] = \frac{1}{2^N} < O\left(\frac{1}{N^2}\right)$. Thus, with probability at least $\Pr[1 \leq X \leq \frac{3N}{4}] = 1 - \Pr[X = 0] - \Pr[X \geq \frac{3N}{4}] > 1 - O(\frac{1}{N^2})$, the number of stations is reduced by a factor of at least $\frac{3}{4}$.

Let N_t, $(t \geq 1)$, denote the number of stations remaining at the beginning of the t-th round and let T be the integer satisfying the double inequality

$$N_{T+1} < \frac{n}{\log n} \leq N_T. \tag{5}$$

Then, for all t, $(1 \leq t \leq T)$, the inequality

$$N_{t+1} \leq \frac{3N_t}{4} \tag{6}$$

holds with probability at least $1 - O(\frac{1}{{N_t}^2}) > 1 - O((\frac{\log n}{n})^2)$.

Suppose that for all t, $(1 \leq t \leq T)$, the inequality (6) holds. Then, $N_t \leq \left(\frac{3}{4}\right)^{t-1} n$ holds. Hence, the inequality (5) guarantees $T \in O(\log \log n)$. Thus, the probability that (6) holds for all t, $(1 \leq t \leq T)$, is at least $1 - O(T(\frac{\log n}{n})^2) > 1 - O(\frac{1}{n})$. Further, in this case, the work complexity of the first T rounds in the protocol is

$$\sum_{t=1}^{T} N_t \leq \sum_{t=1}^{T} \left(\frac{3}{4}\right)^{t-1} n = O(n).$$

We just showed that with probability at least $1-O(\frac{1}{n})$, the work complexity of the first T rounds is bounded by $O(n)$ and that at most $N_{T+1} \leq \frac{n}{\log n}$ stations remain at the end of the T-th broadcast round. Now, by virtue of Theorem 2, after the $(T+1)$-th round, the leader is elected in $16\lceil \log n \rceil - T$ rounds with probability $1-O(\frac{1}{n})$. Hence, in this case, the work complexity of the $16\lceil \log n \rceil - T$ rounds is $(16 \log n - T) \times \frac{n}{\log n} \in O(n)$. To summarize our findings, we state the following important result.

Theorem 3. *With probability at least* $1 - O(\frac{1}{n})$*, the work complexity of the protocol* `Logarithmic-leader-election` *is bounded by* $O(n)$.

3.2 Electing a Leader in Doubly-Logarithmic Time

This subsection is devoted to presenting the details of an $O(\log \log n)$-round randomized leader election protocol that works under the conditions of Scenario 3, that is, when the number n of stations is not known beforehand. The protocol is reminiscent of the one developed in the previous subsection. The main difference is that the probability of a station to exit the protocol varies in each round, and is determined by a variable m, as we are about to specify. The details follow.

```
Protocol Doubly-logarithmic-leader-election
each station sets $m \leftarrow 0$;
repeat
    each station broadcasts on channel C(1) with probability $\frac{1}{2^{2^m}}$;
    if the status of channel C(1) is NULL then
        each station sets $m \leftarrow \max\{m-1, 0\}$
    endif
    if the status of channel C(1) is COLLISION then
            all stations that did not broadcast in this round quit the protocol;
            all the remaining stations set $m \leftarrow m+1$;
    endif
until the status of channel C(1) is SINGLE;
the station that has broadcast in the last round is declared the leader.
```

Clearly, upon termination, the protocol `Doubly-logarithmic-leader-election` correctly returns a leader. To assess the expected behavior of the protocol, we now estimate the number of rounds necessary for the protocol to terminate. Suppose that at the beginning of a given round there are N stations remaining and that the probability that a station broadcasts is p. We shall assume, for the sake of simplicity, that the number of stations broadcasting during this round is exactly the expected number of stations to broadcast. In other words, we assume that in each round

$$\text{out of the } N \text{ remaining stations, exactly } Np \text{ stations broadcast.} \quad (7)$$

Let T be the integer satisfying

$$2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdots 2^{2^{T-2}} < n \leq 2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdots 2^{2^{T-1}}. \quad (8)$$

It is easy to see that (8) can also be written as: $2^{2^{T-1}-1} < n \leq 2^{2^{T}-1}$ confirming that $T \in \Theta(\log\log n)$.

For $t \geq 1$, let N_t denote the number of stations remaining at the beginning of the t-th round of the protocol. Assuming that n and T are large enough, we will evaluate each N_t. Assumption (7) guarantees that in the first round out of n stations $\frac{n}{2^{2^0}} = \frac{n}{2}$ stations broadcast. Thus $N_2 = \frac{n}{2^{2^0}}$ stations remain at the beginning of the second round.

In the second round $m = 1$ and by assumption (7) $\frac{N_2}{2^{2^1}} = \frac{n}{2^{2^0} \cdot 2^{2^1}}$ stations broadcast. Thus $N_3 = \frac{n}{2^{2^0} \cdot 2^{2^1}}$ stations remain at the beginning of the third round.

Continuing in this way, $N_t = \frac{n}{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \ldots 2^{2^{t-2}}}$ stations remain at the beginning of the t-th round, $(1 \leq t \leq T)$, and $\frac{N_t}{2^{2^{t-1}}} = \frac{n}{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \ldots 2^{2^{t-1}}}$ stations broadcast. By (8), in the T-th round $m = T - 1$ and $\frac{N_T}{2^{2^{T-1}}} = \frac{n}{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \ldots 2^{2^{T-1}}} \leq 1$ stations broadcast, and the status of C(1) is NULL or SINGLE. If exactly one station broadcasts, the protocol terminates and a leader is elected. Otherwise, the status of channel C(1) is NULL and no station quits. Further, m is decreased to $T - 2$, and

$$N_{T+1} = N_T = \frac{n}{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdots 2^{2^{T-2}}}. \tag{9}$$

Now (8) and (9) combined allow us to write

$$N_{T+1} \;\leq\; \frac{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdots 2^{2^{T-1}}}{2^{2^0} \cdot 2^{2^1} \cdot 2^{2^2} \cdots 2^{2^{T-2}}} \;\leq\; 2^{2^{T-1}}. \tag{10}$$

In the $(T+1)$-th round, $m = T - 2$ and, by assumption (7), $\frac{N_{T+1}}{2^{2^{T-2}}}$ stations broadcast. If the status of channel C(1) is NULL, that is, if $N_{T+1} < 2^{2^{T-2}}$, then m is decreased by one. If the status of channel C(1) is SINGLE, a leader is elected and the protocol terminates. Therefore, we may assume that the status of channel C(1) is COLLISION. In this case, $N_{T+2} = \frac{N_{T+1}}{2^{2^{T-2}}}$ stations remain. Now, (10) guarantees that

$$N_{T+2} \;\leq\; \frac{N_{T+1}}{2^{2^{T-2}}} \;\leq\; 2^{2^{T-2}}. \tag{11}$$

In the $(T+2)$-th round $m = T - 1$ and, by assumption (7), $\frac{N_{T+2}}{2^{2^{T-1}}}$ stations broadcast. Note that equation (11) guarantees that the status of channel C(1) must be NULL, no station quits in this round, and $N_{T+3} = N_{T+2}$.

In the $(T+3)$-th round $m = T - 2$ and, by assumption (7), $\frac{N_{T+2}}{2^{2^{T-2}}}$ stations broadcast. If (11) holds with equality, then exactly one station broadcasts and the protocol terminates. Otherwise, the status of the channel must be NULL, m is again decreased by one, and $N_{T+4} = N_{T+2}$.

In the $(T+4)$-th round, $m = T - 3$, and by assumption (7), $\frac{N_{T+2}}{2^{2^{T-3}}}$ stations broadcast. Note that by (11) it must be the case that

$$N_{T+5} \;=\; \frac{N_{T+2}}{2^{2^{T-3}}} \;\leq\; 2^{2^{T-3}}. \tag{12}$$

Now, (11) and (12) combined imply that three rounds, namely, $T+2$, $T+3$ and $T+4$, have reduced the number of stations from $2^{2^{T-2}}$ to $2^{2^{T-3}}$. This argument shows that $O(T)$ rounds suffice to elect a leader. Since $T \in \Theta(\log\log n)$, we have the following result.

Lemma 2. *Under the assumption that in each round the expected number of stations broadcast, the protocol* `Doubly-Logarithmic-leader-election` *elects a leader among n stations in $O(\log\log n)$ rounds.*

Since each station decides to broadcast independently, assumption (7) is not entirely realistic. As it turns out, a formal proof of the fact that the protocol terminates in $O(\log\log n)$ rounds without this assumption is much more difficult and, due to stringent page limitations, will only appear in the journal version of this paper. Nonetheless, we state this important result here.

Theorem 4. *The protocol* `Doubly-Logarithmic-leader-election` *elects a leader among n stations in $O(\log\log n)$ rounds with probability at least $1 - O(\frac{1}{\log n})$.*

Theorem 5. *The work complexity of the protocol* `Doubly-Logarithmic-leader-election` *is $O(n)$ with probability at least $1 - O(\frac{1}{n})$.*

Although Theorem 4 guarantees that `Doubly-logarithmic-leader-election` has a very appealing "doubly-logarithmic" performance, it is certainly of interest to see how the protocol performs in practice. We refer the reader to Table 1 where a number of simulation results are summarized. These results show that our protocol has an excellent practical performance, featuring a low proportionality constant. The protocol was simulated 1000 times for each number of stations in the first row of Table 1. The second row features the average number of rounds needed by the protocol to elect a leader. It is very encouraging to note that even in the case of 100 million stations, the average number of rounds required by our protocol to elect a leader is only 10.7.

Table 1. Simulation of `Doubly-logarithmic-election`

# stations	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8
# rounds	3.8	5.4	7.2	7.9	8.6	9.6	10.1	10.7

References

1. J. Bentley and A. Yao, An almost optimal algorithm for unbounded search, *Information Processing Letters*, 5, (1976),:82–87. 211
2. D. Bertzekas and R. Gallager, *Data Networks*, Second Edition, Prentice-Hall, 1992. 209, 210

3. R. Bar-Yehuda, O. Goldreich, and A. Itai, Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection, *Distributed Computing*, 5, (1991), 67–71. 209, 210
4. J. I. Capetanakis, Tree algorithms for packet broadcast channels, *IEEE Transactions on Information Theory*, IT-25, (1979), 505–515. 210
5. I. Chlamtac and S. Kutten, Tree-based broadcasting in multihop radio networks, *IEEE Transaction on Computers*, C-36, (1987), 1209–1223. 210
6. J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, 1992. 212
7. R. M. Metcalfe and D. R. Boggs, Ethernet:distributed packet switching for local computer networks, *Communications of the ACM*, 19, (1976), 395–404. 210, 211, 212
8. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995. 212
9. D. E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM Journal on Computing*, 15, (1986), 468–477. 209, 211

Random Regular Graphs with Edge Faults: Expansion through Cores

Andreas Goerdt

TU Chemnitz, Theoretische Informatik
D-09107 Chemnitz, Germany
goerdt@informatik.tu-chemnitz.de

Abstract. Let G be a given graph (modeling a communication network) which we assume suffers from static edge faults: That is we let each edge of G be present independently with probability p (or absent with *fault probability* $f = 1-p$). In particular we are interested in robustness results for the case that the graph G itself is a random member of the class of all regular graphs with given degree d.
Here we deal with expansion properties of faulty random regular graphs and show: For $d \geq 42$, fixed and $p = \kappa/d$, $\kappa \geq 20$, a random regular graph with fault probability $f = 1-p$ contains a linear-sized subgraph which is an expander almost surely. This subgraph can be found by a simple linear-time algorithm.

Introduction

Modern multiprocessor architectures and communication networks compute over structured interconnection graphs like meshes. Here several applications share the same network while executing concurrently. This may of course lead to unavailability of links and nodes in certain cases and we may assume to compute over a subnetwork being randomly assigned by the operating system. Moreover, this subnetwork may suffer from edge or node faults. Our work addresses robustness properties in case the subnetwork is a random regular graph suffering from edge faults. Random regular graphs make an (at least theoretically) popular choice because they combine low degree with high expansion almost always (see [AlSp 92] for an introduction to expansion and [Bo 85] to random regular graphs). In case of structured networks like the butterfly or the hypercube it is known how to simulate the non-faulty network on the faulty one with a well-determined slowdown, for example [Le et al. 92] ,[Ta 92] . In [Ka et al. 94] random graph concepts and techniques are applied to the butterfly.
Our study continues the work begun in [Ni et al. 94], [NiSp 95, Go 97] which are the only papers known to us which investigate random regular graphs with edge faults. The paper [Ni et al. 97] contains an application of results on faulty random regular graphs to fat trees, a class of interconnection networks with universal simulation properties. In [Go 97] we show that $p = 1/d - 1$ is a threshold probability for the existence of a linear- sized component in faulty random regular graph. Note that a linear- sized component is an absolute necessity in order

to simulate the non-faulty network on the faulty one with only constant slowdown (Slowdown is determined as the fraction of the time of the simulation on the faulty network and the time needed on the non-faulty network.) But, to achieve only constant slowdown, we need more than just a linear-sized component: We need (at least) a linear-sized subgraph which preserves the expansion properties (cf. fact 1) which are crucial for efficient communication in non-faulty random regular graphs. Recently, see [Ku 97], processing networks with faulty edges have been investigated from the point of view of message routing. In this context faultiness corresponds to congestion of an edge.
The interesting paper [NiSp 95] shows that the first eigenvalue of the (or "a" or "any", the uniqueness is not really known) linear-sized component of a faulty random regular graph is bounded away from the average degree. This implies good expansion properties, but does not directly give us an expanding subgraph, necessary for the above mentioned simulation. In section 1 we show, that the linear-sized component itself is no expander. (Note: component is a *maximal* connected subgraph.) This motivates our work to actually find a linear-sized expanding subgraph. In section 2 we present a simple edge deletion process which will give us a linear-sized subgraph which is an expander with high probability if the fault probability is not too high. Our edge deletion process iteratively deletes all edges incident with nodes of degree ≤ 2. It thus finds the 3-core (=unique maximal subgraph where each node has degree ≥ 3) of the faulty graph. Due to its randomness properties this core is an expander with high probability. In the subsequent sections 3, 4 a probabilistic analysis of this algorithm is performed. It is inspired by the analysis in [BrFrUp 93] of a simple algorithm for 3-satisfiability. The tight analysis of an edge deletion process similar to ours for random graphs without degree bound [PiSpWo 96] cannot directly be transferred to the present situation because the crucial differential equations (5.7) need to be modified somehow to reflect the degree bound d.
In view of [PiSpWo 96] the intuition of our analysis is: In general random graphs with edge probability c/n, c a constant, the degree of a given node is approximated by the Poisson distribution with mean c.When c is large enough we get a linear-sized 3-core. In the case of faulty random regular graphs we get the binomial distribution with parameters d, $p = \kappa/d$. for the degree of a given node. The Poisson distribution is the limit of this binomial distribution when d gets large. Full proofs can be found in [Go 98].

1 Random Regular Graphs and Configurations

The probability space of random regular graphs with edge faults is given by the following probabilistic generation procedure: 1. Choose a d-regular graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ according to the uniform distribution. To ensure the existence of such graphs we will always assume that $d \cdot n$ is even. 2. Delete randomly K edges from G. Each set of K edges is equally likely to be deleted. For the rest of this paper we make the following notational conventions: The degree d of our graphs is fixed. The number of nodes is n. We assume that n

gets large, and is such that nd is even. We let $N = nd/2$. The number of non-faulty edges is $L = \kappa/d \cdot dn/2 = \kappa \cdot n/2$, where κ is a constant independent of d. We always assume $d \geq \kappa$. Then $K = N - L$ is the number of faulty edges. The standard tool to deal with random regular graphs are random configurations ([Bo 85], p. 47ff). Random regular graphs with edge faults are dealt with by random configurations with edge faults [Ni et al. 94]: Here we have n disjoint sets $W_1, \ldots, W_n$ where each W_i consists of d elements. We let always $\mathcal{W} = W_1 \,\dot{\cup}\, \ldots \,\dot{\cup}\, W_n$, the W_i are called classes, the elements of the W_i are nodes, sets $\{v, w\}$ where $v, w \in \mathcal{W}$, $v \neq w$ are called edges. A configuration is a partition of $\mathcal{W}$ into $N = dn/2$ edges. We have $(2N - 1)!! = (2N-1) \cdot (2N-3) \cdots 3 \cdot 1 = (2N)!/(N! \cdot 2^N)$ configurations. A random configuration with edge faults is given by the following probabilistic procedure: 1. Choose a configuration Γ from the uniform distribution. 2. Delete randomly a set $\Delta \subseteq \Gamma$ of K edges from Γ. We represent this procedure as a probability tree where each leaf has the probability $1/(2N-1)!! \cdot 1/\binom{N}{K}$. We mark each leaf of the tree with the corresponding set Φ of $L = N - K$ non-faulty edges. The probability of Φ is the sum of the probabilities of the leaves representing Φ. We call this probability space $\mathrm{Con}(L, n)$. To show that a property holds for almost all faulty graphs, i. e. simply with probability tending to 1, we can show the analogous result for faulty configurations (cf. [Bo 85], p. 47 ff).
We use the following notions of high probability: "Almost surely" or "with high probability" or "almost all" means that the probability goes to 1 when n goes to infinity. "Quite surely" means, for any constant $a > 0$ the probability is $\geq 1 - O(1/n^a)$ for all sufficiently large n.
A slightly different notion of faulty random regular graphs and configurations is considered in [Go 97, Ni et al. 94, NiSp 95], where step 2 of our probabilistic generation procedure is replaced by:2. Delete each edge independently with probability $1 - \kappa/d$. These models are essentially equivalent (cf. the analogous situation for random graphs [Bo 85], p.33). The following fact is from [Bo 88] (part(a)) and from [Go 97] (part(b)).

Fact 1 (a) There is a constant (independent of d and n) $c > 0$ such that for a random configuration Γ the following holds with high probability: For all subsets X of classes we have $| \mathcal{N}(X) | \geq c \cdot \mathrm{Min}\{| X |, | \mathrm{Cpl}\ X |\}$, where $\mathcal{N}(X)$ is the set of classes adjacent to X in Γ but not belonging to X. Hence, random configurations and random regular graphs are (c-)expanders, with high probability.The maximal such c is called the expansion constant of Γ.
(b) If $p = \kappa/d > 1/(d-1)$ then the following holds in a faulty random regular graph (proved in the independent edge deletion model above): There is a constant $\varepsilon = \varepsilon(\kappa)$, such that almost surely a faulty configuration has a connected component having at least $\varepsilon \cdot n$ classes.

Theorem 2 shows that there exists a linear-sized component which as a whole is no expander. Note that we do not know yet that we have only 1 linear-sized component. (However, this is likely by random graph experience [Ka et al. 94, AlSp 92]).

Theorem 2. Let k be an arbitrary constant. With high probability a faulty random configuration where $1 > p = \kappa/d > 1/(d-1)$ has a linear- sized component, whose expansion constant is $\leq 1/k$

Proof: We employ the "double randomization trick", e. g. [Go 97]. Let $\varepsilon > 0$ be small enough such that $p' = p - \varepsilon = \frac{\kappa'}{d} > \frac{1}{d-1}$. Let

$$p'' = 1 - \frac{1-p}{1-p+\varepsilon} > 0 \text{ then } (1-p') \cdot (1-p'') = 1-p \ \ (\text{note } p'' < 1\ ).$$

Hence, when we modify step 2 of our generation procedure such that we first throw the edges (of the underlying random configuration) with p' and then with p'' we get a faulty random configuration with p. Our proof follows these 2 probabilistic experiments and consists of 2 steps:
Step 1: After throwing with probability p' we have with high probability a linear-sized component C and a linear number of *isolated* paths each of length k. We denote by $\mathcal{P}$ the collection of these paths. Step 2: After throwing the edges with p'' we have with high probability at least 1 path $P \in \mathcal{P}$ connected with C via a path consisting of edges thrown in by experiment 2. Moreover, P may be considered as a fringe of C: The path P has only 1 neighbour reachable via a non faulty edge.
Step 1 is shown by running sufficiently many (breadth-first) searches on disjoint parts of a faulty configuration. With probability bounded away from 0 (when n gets large) an isolated path of length k is found by a single search. Tail bounds imply the almost sure existence of a linear number of such paths. We formalize our search algorithms as (breadth-first) generation algorithms as in [Go 97].Our algorithm directing the single searches uses the following global variables. These are implicitly updated each time they change: E = the set of non-faulty edges generated. $\neg E$ = the set of faulty edges generated. Free = the set of *nodes* which have not been looked at, that is which do not occur in $E \cup \neg E$. $\neg$ Free = the set of nodes which have already been looked at. Dis = the set of *classes* which are incident with faulty or non-faulty edges generated by previous searches. Path = the set of classes discovered via non-faulty edges by the current search. We fix a constant γ, sufficiently small. The single searches are called from the procedure Gen:

```
Procedure Gen
1. For $i = 1$ to $\gamma \cdot n/k$ do
2.         Pick a class $S \notin$ Dis deterministically; Path := $\emptyset$
3.         Search($S$); Dis := Dis $\cup$ Path
```

The variable Path in the subsequent procedure Search(S) is used as follows: Path will always contain what can be looked at as part of an isolated path consisting of k classes. If Path contains a different structure, the current Search(S) stops with an error. The interior classes of this path contain no node from Free. Hence these classes are incident with sufficiently many faulty edges in order to make Path isolated. A class added as new end class to Path will contain only nodes from Free. This is important to bound the error-probability of Search(S) away from 1.

```
Procedure Search($S$)
1. Path := $\{S\}$
2. while |Path| $\leq k$ and Path has a class $T$ at an end with $|T \cap \text{Free}| > 0$
3.        Pick such a class $T$ deterministically.
4.        while $|T \cap \text{Free}| > 0$ do
5.            Pick a node $x \in T \cap$ Free deterministically.
6.            Choose $y \in \text{Free} \setminus \{x\}$ from the uniform distribution.
              Add the edge $(x, y)$ to $E$ with probability $p'$, to $\neg E$ with $1 - p'$.
7.            if $(x, y) \in E$ then
                  Add the class of $y$, we call it $R$, to Path. Return
                  with error, if $|R \cap \text{Free}| < d - 1$ or path cannot any
                  longer be extended to an *isolated* path of $k$ classes due to $R$.
8.            if $(x, y) \in \neg E$ and this implies that Path
              cannot be extended as required, then return with error.
```

A successful return of Search(S) occurs, when the while-loop from 2. is left regularly. This happens for a given call of Search(S) inside Gen with probability bounded away from 0. To this end it is helpful to visualize each computation of Gen as a path in a probability tree. Conditioning on an arbitrary history H of Gen which ends just before a new execution of Search(S) we get that Prob[Search(S) ends without an error] $\geq \varepsilon(d, \kappa', \gamma, k) > 0$. Hence the expected number of successful returns from Search(S) is at least $(\gamma \cdot n/k) \cdot \varepsilon$ which is linear in n (but decreasing in k and d). Tail bounds for the binomial distribution imply that we get quite surely a linear (in n) number of isolated paths in our set P and step 1 is finished. We show step 2: We condition everything what follows on the almost sure event that the underlying configuration is an expander. In the underlying non-faulty configuration we define $C_0 = C$ and $C_{i+1} = \mathcal{N}(C_i)$ (cf. fact 1). The expansion property implies that there is a j depending on $|C|$ and $|\mathcal{P}|$ the number of classes in C and $\mathcal{P}$, such that C_j and $\mathcal{P}$ have $\geq 1/2 \cdot |\mathcal{P}|$ many classes in common. This is simply because C_j has $n \cdot (1 - (1/2)|\mathcal{P}|)$ many classes . As j is constant, we have a linear in n number of classes W in $\mathcal{P}$ over which $\mathcal{P}$ is entered. That is we have a path from C to W which does not hit $\mathcal{P}$ before. As k is fixed, we have a linear number of paths in $\mathcal{P}$, $P_1, \ldots, P_{\alpha n}$ which contain such an entering class W. Throwing in the edges of the underlying configuration with p'' gives us a linear number of P_i connected to C in the required way (with high probability) using the "entering" paths.

2 The Edge Deletion Process

For a node $x \in \mathcal{W}$, $\text{Deg}_{\Phi}(x) = \text{Deg}_{\Phi}(W)$, the number of nodes in the class W incident with non-faulty edges and W is the class to which x belongs. The 3-core of Φ is the maximal subconfiguration Φ' such that $\text{Deg}_{\Phi'}(x) \geq 3$ for each x from Φ'. The following edge deletion process iteratively strips away the edges incident with classes of degree ≤ 2.

```
Input: A faulty configuration $\Phi \in \mathrm{Con}(L, n)$.
Output: The 3-core of $\Phi$.
while $\Phi$ has classes of degree $\leq 2$ do
        $\Phi := \Phi \setminus \{\{x, y\} | \mathrm{Deg}_\Phi(x) \leq 2$ or $\mathrm{Deg}_\Phi(y) \leq 2\}$
od;     return $\Phi$ as the 3-core.
```

For our probabilistic analysis of this algorithm we use the probability spaces $\mathrm{Con}(m, \bar{n}) = \mathrm{Con}(m, n_0, n_1, n_2, n_3)$ where $n = n_0 + n_1 + n_2 + n_3$, which contains as atoms the faulty configurations with exactly: m edges, n classes of which are: n_0 of degree 0, n_1 of degree 1, n_2 of degree 2, and n_3 of degree ≥ 3. Each configuration is equally likely. When considering $\mathrm{Con}(m, \bar{n})$ we use the following names and abbreviations: A class is heavy in $\Phi \in \mathrm{Con}(m, \bar{n})$ if it is of degree ≥ 3 in Φ, it is light if its degree is 1 or 2. A node is heavy or light, iff it belongs to a heavy or light class. We let

$$l = n_1 + 2n_2, \ h = 2m - l, \ \text{ and } \lambda = \frac{2m - l}{n_3} = \frac{h}{n_3}.$$

Hence, l is the number of light nodes, h the number of heavy nodes and λ is the average degree of a class W given that W is heavy. Note $d \geq \lambda \geq 3$. When we run our edge deletion process with $\Phi \in \mathrm{Con}(m, \bar{n})$ we let Φ_i be the configuration obtained from Φ after i rounds of the loop ($\Phi_0 = \Phi$). Moreover we have m_i, $\bar{n}_i$, $n_{0,i}$, $n_{1,i}$, $n_{2,i}$, $n_{3,i}$, l_i and λ_i for $i \geq 0$ as random variables on $\mathrm{Con}(m, \bar{n})$. For example $n_{0,i}(\Phi) = n_0(\Phi_i) =$ the number of classes of degree 0 in Φ_i. As usual when a probabilistic analysis is possible, the (uniform) distribution needs to be preserved.

Lemma 1. (a) We consider $\mathrm{Con}(m,\bar{n})$ and $\mathrm{Con}(m',\bar{n}')$. For Θ, $\Theta' \in \mathrm{Con}(m', \bar{n}')$ we have: $|\{\Phi \in \mathrm{Con}(m, \bar{n}) | \Phi_1 = \Theta\}| = |\{\Phi \in \mathrm{Con}(m, \bar{n}) | \Phi_1 = \Theta'\}|$, that is each configuration from $\mathrm{Con}(m', \bar{n}')$ is hit by the same number of configurations from $\mathrm{Con}(m, \bar{n})$ after 1 round of edge deletion.
(b) For $\Theta, \Gamma \in \mathrm{Con}(m, \bar{n})$ we have for a random $\Phi \in \mathrm{Con}(L, n)$ and $i \geq 0$ fixed: $\mathrm{Prob}[\Phi_i = \Theta] = \mathrm{Prob}[\Phi_i = \Gamma]$. Hence, we have in the space $\mathrm{Con}(L, n)$: Conditional on the event that $\Phi_i \in \mathrm{Con}(m, \bar{n})$, each configuration from $\mathrm{Con}(m, \bar{n})$ is equally likely.

We need to determine the probability distribution of the degree of a fixed class of degree ≥ 3 in $\mathrm{Con}(m, \bar{n})$. In general, i. e. when the degree is unbounded, we have the concept of Poissonization in this situation [BrFrUp 93, PiSpWo 96]. The present situation leads to the concept of "binomialization":

Lemma 2. We consider the probability space $\mathrm{Con}(m, \bar{n})$, conditional on the event that $W_1, \ldots, W_{n_3}$ are the actual heavy classes. Let $0 < \rho \leq 1$ and $Y = Y(\rho) \overset{D}{=} [\mathrm{Bin}(d, \rho) | \mathrm{Bin}(d, \rho) \geq 3]$, where $\overset{D}{=}$ means equality in distribution and $\mathrm{Bin}(d, \rho)$ is the binomial distribution with parameters d and ρ. Let $(X_1, \ldots, X_{n_3})$ be the random vector of the degrees of our heavy classes $W_1, \ldots, W_{n_3}$.
(a)Let $Y_1, \ldots, Y_{n_3}$ be i.i.d. with $Y_i \overset{D}{=} Y$. Then
$(X_1, \ldots, X_{n_3}) \overset{D}{=} [(Y_1, \ldots, Y_{n_3}) | \sum Y_i = h]$. The right-hand side of the equation

means $(Y_1, \ldots, Y_{n_3})$ conditional on the event that $\sum Y_i = h$, the number of heavy nodes.
(b) The function $EY = (EY)(\rho) = E[Y(\rho)]$ is strictly increasing for $0 < \rho \leq 1$. For λ with $3 < \lambda \leq d$ we have a unique $\hat{\rho}$ such that
$(EY)(\hat{\rho}) = \lambda = E[X_i]$.

Some more abbreviations: When considering $\mathrm{Con}(m, \bar{n})$ where $3 < \lambda \leq d$ we use the following parameters: $\hat{\rho} \in (0,1]$ is uniquely given by:
$\lambda = \mathrm{E}[\mathrm{Bin}(d, \hat{\rho}) | \mathrm{Bin}(d, \hat{\rho}) \geq 3]$. Moreover $\hat{\lambda} = \hat{\rho} \cdot d$. Hence $\hat{\rho}$ causes the Y_i to have the right expectation. Note, $0 < \hat{\lambda} \leq d$ for $\hat{\rho} \in (0,1]$ and $\lambda > \hat{\lambda}$ for $\hat{\rho} \in (0,1)$ but $\lambda = \hat{\lambda} = d$ for $\hat{\rho} = 1$. As usual in the present context $\hat{\lambda}_0$, $\hat{\lambda}_1, \ldots$ are considered as random variables on $\mathrm{Con}(m, \bar{n})$, the index indicating the number of rounds of edge deletion (of course $\hat{\lambda}_0$ is constant, but for $i > 0$ the $\hat{\lambda}_i$'s depend on the actual Φ).
Concerning the probability distribution of the degree of a fixed heavy class W we have the following statement which follows from the definition of the Y_i:

$$\mathrm{Prob}[\mathrm{Deg}_{\Phi}(W) = k] = \frac{\mathrm{Prob}[\mathrm{Y}_1 = k] \cdot \mathrm{Prob}[\mathrm{Y}_2 + \cdots + \mathrm{Y}_{n_3} = h - k]}{\mathrm{Prob}[\mathrm{Y}_1 + \cdots + \mathrm{Y}_{n_3} = h]},$$

The local limit theorem for sums of lattice-type random variables [Du 91, Fe 71], allows us to estimate the fraction $\mathrm{Prob}[\mathrm{Y}_2 + \cdots + \mathrm{Y}_{n_3} = h - k])/(\mathrm{Prob}[\mathrm{Y}_1 + \cdots + \mathrm{Y}_{n_3} = h])$, as $1 + o(1)$ (for $n_3 \to \infty$) provided the ρ from lemma 4(b) is chosen as $\hat{\rho}$. In this case we have $h = E[\sum Y_i]$, see also [BrFrUp 93], [PiSpWo 96] for a similar application of the local limit theorem.

Corollary 1. We consider the probability space $\mathrm{Con}(m, \bar{n})$ conditional on the event that the class W is heavy. We let n_3 go to infinity and assume that λ is bounded away from 3 and d. Then we have the following estimate:

$$\mathrm{Prob}[\mathrm{Deg}_{\Phi}(W) = k] \leq \frac{(1-\hat{\rho})^d}{B(\hat{\rho},3)} \frac{1}{k!} \left(\frac{\hat{\lambda}}{1-(\hat{\lambda}/d)} \right)^k,$$

where $B(\hat{\rho}, 3)$ is the probability that the binomial random variable with parameters d and $\hat{\rho}$ is ≥ 3.

3 One Round of Edge Deletion

We use some more abbreviations:

$$\eta = \frac{(1-\hat{\rho})^d}{B(\hat{\rho},3)}, \ \mu = \frac{\hat{\lambda}}{1 - \frac{\hat{\lambda}}{d}}, \ \nu = \frac{l}{2m}, \ \omega = \nu \cdot \mu.$$

Note that the values depend on the actual probability space $\mathrm{Con}(m, \bar{n})$ considered. We consider $\eta_0, \eta_1, \ldots, \mu_0, \mu_1, \ldots$ as random variables on $\mathrm{Con}(m, \bar{n})$, the index indicating the number of rounds of edge deletion. Moreover, δ with $1 > \delta > \frac{1}{2}$ is fixed. Of particular importance is ν, the proportion of light nodes among all nodes still under consideration.

To calculate probabilities in $\mathrm{Con}(m,\bar{n})$ it is helpful to consider each $\Phi \in \mathrm{Con}(m,\bar{n})$ as generated by the following string generation procedure. Note that each configuration is actually generated $m! \cdot 2^m$-times: 1. Choose 2m nodes from $\mathcal{W}$ according to the degree constraints given by $\bar{n}$. Each single possibility is equally likely. 2. Choose a random permutation of the nodes chosen in 1. We visualize this as putting the nodes chosen in 1. randomly into $2m$ slots: first slot 1, then slot 2, $\ldots$, slot $2i-1, 2i$ represent 1 edge. Each possibility has the probability $\frac{1}{(2m)!}$. The following lemma 6 follows by analyzing the above generation procedure, the sharp concentration result follows from tail estimates for martingales (Azuma's inequality), [AlSp 92], chapter 7. See [BrFrUp 93] for the application of Azuma's inequality in a similar context.

Lemma 3. Let $\varepsilon, \varepsilon' > 0$. We consider the probability spaces $\mathrm{Con}(m,\bar{n})$ where $0 < \varepsilon \leq \hat{\rho} \leq 1-\varepsilon < 1$. (This restriction implies that $\hat{\lambda}$ is bounded away from 0 and d, and therefore η and μ can be bounded above by a constant.) We assume that $n_3 \geq \varepsilon' \cdot n$. After 1 round of edge deletion we have:

(a) $\mathrm{E}[m_1] = m(1-\nu)^2(1+O(\frac{1}{m}))$.

(b) $\mathrm{E}[n_{0,1} - l - n_0] \leq n_3\eta \cdot \sum_{k\geq 3} \frac{1}{k!}\omega^k(1+O(\frac{1}{m}))$. (Note that $n_{0,1} - l - n_0$ is the number of heavy classes which get degree 0 after 1 round of edge deletion.)

(c) $\mathrm{E}[n_{1,1}] \leq n_3\eta\mu \cdot \sum_{k\geq 2} \frac{1}{k!}\omega^k(1+O(\frac{1}{m}))$

(d) $\mathrm{E}[n_{2,1}] \leq \frac{1}{2}n_3\eta\mu^2 \cdot \sum_{k\geq 1} \frac{1}{k!}\omega^k(1+O(\frac{1}{m}))$

(e) $\mathrm{E}[l_1] \leq \mathrm{E}[n_{1,1}] + 2\mathrm{E}[n_{2,1}]$

Moreover, for all these random variables we have the same sharp concentration result as for m_1: Quite surely (in $\mathrm{Con}(m,\bar{n})$) $|m_1 - \mathrm{E}[m_1]| \leq n^\delta$.

First we bring lemma 6, in particular the sharp concentration part, into a more manageable form. To this end we fix δ' with $\delta' > \delta$ and assume that the number of light nodes l is not to small, i. e. $l \geq n^{\delta'}$. This implies that we have for the deviation n^δ from the sharp concentration result that $n^\delta = o(l)$, making the following estimates possible:

Theorem 3. We consider $\mathrm{Con}(m,\bar{n})$ and make 1 round of edge deletion. We let $\varepsilon > 0$ be fixed and assume $n_3 = n_{3,0} \geq \varepsilon \cdot n$, hence $m \geq \varepsilon \cdot n$. We make the following additional assumptions about $\mathrm{Con}(m,\bar{n})$: $d/2 > \lambda \geq 16$, $\nu \leq 1/10$, $\omega \leq 3/4$, $l \geq n^{\delta'}$. We have quite surely (with respect to $\mathrm{Con}(m,\bar{n})$) after 1 round of edge deletion:
(a) $m_1 = m \cdot (1-\nu)^2 + O(n^\delta)$. (b) $n_{0,1} - l - n_0 \leq n_3 \cdot \frac{2}{100} \cdot \nu$.

(c) $n_{1,1} \leq n_3 \cdot \frac{2}{10} \cdot \nu$. (d) $n_{2,1} \leq n_3 \cdot \frac{1}{2} \cdot \nu$.

(e) $n_{3,1} \leq n_3 \cdot (1 - \frac{4}{5} \cdot \nu)$. (f) $\nu_1 \leq \frac{1}{2} \cdot \nu$ (Then $l_1 = \nu_1 \cdot 2m \leq \frac{1}{2}\nu \cdot 2m = \frac{1}{2} \cdot l$. It is important that the *proportion* of light vertices decreases geometrically.)

(g) $\lambda_0 \geq \lambda_1 \geq \lambda_0 \cdot (1 - 3\nu)$. ($\lambda_0$ = the λ of $\mathrm{Con}(m, \bar{n})$.) (h) $\omega_1 \leq \omega$.

Before applying theorem 7 iteratively we look at the beginning applying Azuma's inequality again:

Theorem 4. We consider the probability space $\mathrm{Con}(L, n)$. Let $\tau = \kappa/d$ be the probability that a given edge is present. Quite surely we get:

(a) $n_0 \leq n \cdot \exp(-\kappa)$. (b) $n_1 \leq n \cdot \exp(-\kappa) \cdot \frac{\kappa}{1-\tau}$. (c) $n_2 \leq n \cdot \exp(-\kappa) \cdot \frac{1}{2} \left(\frac{\kappa}{1-\tau}\right)^2$.

(d) $l \leq n \cdot \exp(-\kappa)$ if $\kappa \geq 1$. (e) $n_3 \geq n \cdot (1 - 3\exp(-\kappa))$ if $\kappa \geq 1$.

4 Several Rounds of Edge Deletion

Our edge deletion process is analyzed by an iterative application of lemma 6 and theorem 7. Some intuitive explanation: At first one would possibly try to apply only lemma 6 iteratively: the expectation of the number of light vertices decreases geometrically which should give an expectation of 0 light vertices after logarithmically many rounds of edge deletion. Then Markov's inequality implies the non-existence of light vertices with high probability. But, ν is constant initially and lemma 6(a) does not exclude the possibility of ending up with the empty 3-core after logarithmically many rounds. Moreover the iterated applicability of the lemma cannot be ensured by considering the expectation alone. Therefore we need theorem 7, in particular (f), (g). The other points are necessary to ensure the iterated applicability (quite surely) of the theorem. We use theorem 7 as long as $l \geq n^{\delta'}$. Afterwards we simply use lemma 6 as indicated above. Note that then $v = o(1)$, as m is still linear in n .

Theorem 5. We let $\kappa \geq 20$ (then the number of edges is $L \geq 10 \cdot n$) and $d \geq 42$ fixed. We consider the probability space $\mathrm{Con}(L, n)$:

(a) Quite surely in $\mathrm{Con}(L, n)$ we have: $21 \geq \lambda \geq 19$, $\nu \leq 1/100$, $\omega \leq 1/2$.

(b) Let $j \leq n$, conditional on the event $l_0, l_1, \ldots, l_{j-1} \geq n^{\delta'}$ we have quite surely:
 (i) $\nu_j \leq \left(\frac{1}{2}\right)^j \cdot \nu$. (ii) $\omega_j \leq \left(\frac{1}{2}\right)^{j+1}$.
 (iii) $\lambda \geq \lambda_j \geq \lambda \cdot \prod_{i=0}^{j-1}(1 - 3\nu_i)$. (iv) $n_{3,j} \geq n_{3,0} \cdot \prod_{i=0}^{j-1}(1 - \frac{4}{5}\nu_i)$.

(c) There is a $j = O(\log(n))$ such that we have quite surely: There exists a $k \leq j$ such that $l_k \leq n^{\delta'}$.

Note that (b) (i) and (iv) of theorem 9 imply that the heavy nodes do not disappear. Finally the iterative application of the expectation part of lemma 6 is given by:

Theorem 6. We consider $\mathrm{Con}(m, \bar{n})$ where $l \leq n^{\delta'}$, $\frac{d}{2} \geq \lambda \geq 17$ and $n_3 \geq \varepsilon \cdot n$ for a fixed $\varepsilon > 0$. Let $0 \leq j \leq O(\log(n))$. Conditional on the event that $l = l_0, l_1, \ldots, l_{j-1} \leq n^{\delta'}$, we have:

(a) $\mathrm{E}[l_j] \leq \left(\frac{1}{10}\right)^j \cdot l$. (b) $l_j \leq n^{\delta'}$ quite surely.

(c) $n_{3,j} \geq n_{3,0} - O(j \cdot n^{\delta'})$. (d) $m_j \geq m_0 - O(j \cdot n^{\delta'})$. (e) $\frac{d}{2} \geq \lambda_j \geq 16$.

We now combine theorem 9 and 10 to get the final result: With $\kappa \geq 20$ and $d \geq 42$ the edge deletion process applied to a random member of $\mathrm{Con}(L, n)$ yields with high probability a 3-core of size $\geq \varepsilon \cdot n$, where ε is independent of d. The techniques from [Bo 88] show, that this 3-core is with high probability an expander with expansion constant $\geq \mu \cdot 3/d \cdot 2$ where μ is a not unrealistically small constant < 1.

References

[Ka et al. 94] Anna. R. Karlin, Greg Nelson, Hisao Tamaki. On the fault tolerance of the butterfly. In STOC 1994. 125-133. 219, 221

[Ta 92] Hisao Tamaki. Efficient self-embedding of butterfly networks with random faults. In FoCS 1992. 533 - 541. 219

[Le et al. 92] Tom Leighton, Bruce Maggs, Ramesh Sitaraman. On the fault tolerance of some popular bounded-degree networks. In FoCS 1992. 542-552. 219

[Ni et al. 97] S. Nikiletseas, G. Pantziou, P. Psycharis, P. Spirakis. On the fault tolerance of fat-trees. In Euro-Par 97, Parallel Processing. LNCS 1300. 196-207. 219

[NiSp 95] S. Nikoletseas, Paul Spirakis. Expander properties of random regular graphs with edge faults. In STACS 1995. 421-432. 219, 220, 221

[Ni et al. 94] S. Nikoletseas, K. Palem, P. Spirakis, M. Yung. Short vertex disjoint paths and multiconnectivity in random graphs: Reliable network computing. In ICALP 1994. 508-515. 219, 221

[BrFrUp 93] Andrei Broder, Alan Frieze, Eli Upfal. On the satisfiability and maximum satisfiability of random 3-CNF Formulas. In SoDA 1993. 322-330. 220, 224, 225, 226

[PiSpWo 96] Boris Pittel, Joel Spencer, Nicholas Wormald. Sudden emergence of a giant k-core in a random graph. Journal of Combinatorial Theory, Series B, 67, 1996. 111-151. 220, 224, 225

[Fe 71] William Feller. An introduction to probability theory and its applications II. Wiley, New York, 1971. 225

[Du 91] R. Durrett. Probability: Theory and examples. Wadsworth & Brooks, Pacific Grove, California, 1991. 225

[Go 97] Andreas Goerdt. The giant component threshold for random regular graphs with edge faults. In MFCS 1997.279-288. 219, 221, 222

[Go 98] Andreas Goerdt. Random regular graphs with edge faults: expansion through cores. www.tu-chemnitz.de/informatik/HomePages/TI/Papers/1998/core_l.ps.gz 220

[Bo 88] Bela Bollobas. The isoperimetric number of random regular graphs. European Journal of Combinatorics 9, 1988. 241-244. 221, 228

[Bo 85] Bela Bollobas. Random Graphs. Academic Press, 1985. 219, 221

[AlSp 92] Noga Alon, Joel Spencer. The probabilistic method. Wiley 1992. 219, 221, 226

[Ku 97] Ludek Kucera. Bounds on the throughput of an interconnection network.In SPAA 1997. 139-148. 220

A Quantum Polynomial Time Algorithm in Worst Case for Simon's Problem

(Extended Abstract)

Takashi Mihara[1] and Shao Chin Sung[2]

[1] Department of Electronics and Information Technology
Kyushu Tokai University
9-1-1, Toroku, Kumamoto, 862-8652, Japan
tmihara@ktmail.ktokai-u.ac.jp
[2] School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai Tatsunokuchi Nomi Ishikawa, 923-1292, Japan
son@jaist.ac.jp

Abstract. Simon showed a problem that can be solved in polynomial time on a quantum Turing machine but cannot be solved in polynomial time on any probabilistic Turing machine. However, his quantum algorithm is evaluated in expected time. In this paper, we show a polynomial time quantum algorithm in worst case for Simon's problem. Brassard and Høyer also construct a polynomial time quantum algorithm in worst case for it. However, our algorithm is much simpler than their algorithm.

1 Introduction

Feynman pointed out that current computers may not be able to *efficiently* simulate some quantum physical phenomena because the computational principle of current computers is based on classical physics [7]. Furthermore, he also pointed out that in order to efficiently simulate any physical phenomenon, we will need a new-type computer based on quantum physics, a *quantum computer*. After that, as a model of the quantum computer, Deutsch proposed quantum Turing machines (QTMs) [5], and Bernstein and Vazirani mathematically formalized the QTMs [1,2].

The first indication that a QTM may be more powerful than current computers is Deutsch and Jozsa's result in [6]. They showed a problem that can be solved in polynomial time on a QTM but can not be solved in polynomial time on any deterministic Turing machine (DTM). However, this problem can also be solved in polynomial time on a probabilistic Turing machine (PTM). In 1994, for the first time, Simon showed that QTMs are more powerful than current computers, i.e., he showed a problem that can be solved in polynomial time on a QTM but can not be solved in polynomial time on any PTM [12,13]. His problem is as follows:

Let $f : \{0,1\}^n \to \{0,1\}^m$ with $n \leq m$ be a polynomial time computable function such that for any $x \neq x' \in \{0,1\}^n$ and a nonzero $s \in \{0,1\}^n$,

$$f(x) = f(x') \quad \textit{iff} \quad x' = x \oplus s.$$

Then, find such a nonzero s.

However, since his quantum polynomial time algorithm is evaluated in expected time, we do not satisfy a little. Therefore, we have investigated whether there exists a quantum polynomial time algorithm in worst case for Simon's problem. In this paper, we show a polynomial time algorithm in worst case for it. Brassard and Høyer also construct a polynomial time algorithm in worst case for it, however, their algorithm is more complicated than our algorithm [4]. This is because that their algorithm is combined with Simon's algorithm and Grover's database search algorithm [8]. On the other hand, our algorithm consists of two simple routines (Lemma 1 and Lemma 2) and does not use any probabilistic argument.

2 Quantum Computation

In this section, we briefly review a quantum computation. For more detail, we refer the reader to [1,2,5]. Throughout this paper, let **C** and **Z** be the set of complex numbers and the set of integers, respectively.

Like classical Turing machines, a QTM M also consists of a finite control, an infinite tape, and a tape head. Let Q be the finite set of *states* and Σ the finite set of *tape symbols*. A *configuration* of M is defined by a triple (p, h, α), where $p \in Q$ denotes a current state, $h \in \mathbf{Z}$ denotes an index of the tape cell over which the tape head is currently located, and $\alpha \in \Sigma^*$ denotes a tape content in the tape. Then, the *state transition function* δ of M is

$$\delta : Q \times \Sigma \times \Sigma \times Q \times \{L, R\} \to \mathbf{C}.$$

This transition function $\delta(p, a, b, q, d) = A_m$ means that if M reads a symbol a in a state p, it

1. writes a symbol b on the cell under the tape head,
2. changes the state into q, and
3. moves the head one cell in the direction of $d \in \{L, R\}$.

Further, let c_1 be a configuration before a transition and c_2 a configuration after the transition. Then, the value $|A_m|^2$ is a *transition probability* that M changes its configuration from c_1 to c_2 (the value A_m is called an *amplitude*). Thus, the QTM differs from a classical one in the sense that it evolves on a superposition of configurations and we call a computation using this property a *quantum parallel computation*.

Next, we denote that the state transition function δ corresponds to a *time evolution matrix* (a physical system) U_δ of the QTM M, i.e., each row and column of U_δ corresponds to a configuration of M. Let c_1 and c_2 be two configurations of M. Then, the element corresponding to c_2 row and c_1 column of U_δ is the value of δ evaluated at the tuple that transforms c_1 into c_2 in one step. If no such tuple exists, the corresponding element is zero. Moreover, from physical restrictions, the time evolution matrix U_δ must be a *unitary matrix*, i.e., the relations

$$U_\delta^\dagger U_\delta = U_\delta U_\delta^\dagger = I$$

must be satisfied, where $U_\delta^\dagger$ is the transposed conjugate of U_δ and I is the unit matrix.

Thus, a configuration corresponds to a state vector and a state transition function δ corresponds to a time evolution matrix. Therefore, a computation on a QTM is executed in the following way: let $|\psi_{in}\rangle$ be a vector corresponding to a configuration, $|\psi_{out}\rangle$ a vector corresponding to a next step configuration, and U a matrix corresponding to a state transition function. Then,

$$|\psi_{out}\rangle = U|\psi_{in}\rangle.$$

We also denote it by

$$|\psi_{in}\rangle \xrightarrow{U} |\psi_{out}\rangle.$$

Moreover, the result of a quantum computation is obtained with physical *measurements* (i.e., with observations of the physical systems constructing the QTM) as follows: when a superposition of configurations $|\psi\rangle = \sum_i \alpha_i |c_i\rangle$ is written on the tape, we can measure φ component of ψ with probability $|\langle\varphi|\psi\rangle|^2$. Especially, we can measure c_i component of ψ with probability $|\alpha_i|^2$.

Finally, we denote a quantum bit, the minimum unit of information. A *quantum bit* (or *qubit*) has a chosen *computational basis* $\{|0\rangle, |1\rangle\}$ corresponding to classical bit values 0 and 1 and we denote by

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

For n-qubit ($n \geq 2$), we denote by

$$|x_1, x_2, \ldots, x_n\rangle,$$

where $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. More precisely, it is represented as the tensor products of n qubits.

$$|x_1, x_2, \ldots, x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle (= |x_1\rangle|x_2\rangle \cdots |x_n\rangle),$$

where $\otimes$ is the tensor product operator.

3 A Polynomial Time Algorithm for Simon's Problem

In 1994, Simon showed a problem indicating that QTMs may be more powerful than PTMs, i.e., he showed that there exists a problem that can be efficiently solved on a QTM but can not be efficiently solved on any PTM [12,13].

Simon's problem

Let $f : \{0,1\}^n \to \{0,1\}^m$ with $n \leq m$ be a function such that for any $x \neq x' \in \{0,1\}^n$ and a nonzero $s \in \{0,1\}^n$,

$$f(x) = f(x') \quad \textit{iff} \quad x' = x \oplus s.$$

Then, find such a nonzero s.

Let $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n) \in \{0,1\}^n$. We define $a > b$ by

$$\sum_{i=1}^{n} a_i 2^{i-1} > \sum_{i=1}^{n} b_i 2^{i-1},$$

and we define inner product $a \cdot b$ of a and b modulo 2 by

$$a \cdot b = \sum_{i=0}^{n-1} a_i b_i \pmod 2.$$

Now, we define a transform used in the following lemma. Let H_q be q-dimensional Hadamard matrix ($q = 2^n$), i.e.,

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

and

$$H_q = H_2 \otimes H_{q/2} = \begin{pmatrix} H_{q/2} & H_{q/2} \\ H_{q/2} & -H_{q/2} \end{pmatrix}.$$

Then, the quantum Walsh-Hadamard transform W_{H_q} is as follows:

$$|a\rangle \xrightarrow{W_{H_q}} \frac{1}{q^{1/2}} \sum_{b \in \{0,1\}^n} (-1)^{a \cdot b} |b\rangle.$$

This transform can be executed in $O(n)$ time on a QTM.

We can solve Simon's problem by obtaining a sufficient number of different b's such that $s \cdot b = 0$ (and solves a linear system of equations). Simon's algorithm can obtain a sufficient number of different b's only in polynomial time in expected sense. On the other hand, we show that our algorithm can obtain a sufficient number of different b's in polynomial time in worst case.

First, by Lemma 1, we can obtain a nonzero b such that $s \cdot b = 0$ and $g \cdot b = 1$ for a $g \in \{0,1\}^n$. This lemma means that when we use a different g, we may obtain a different b. Lemma 2 and Lemma 3 certify that we can find a g for obtaining a different b certainly. Thus, our algorithm can certainly find a sufficient number of different b's.

Lemma 1. *Let f be a function that satisfies the condition of Simon's problem. Given an arbitrary nonzero $g \in \{0,1\}^n$, there exists a quantum algorithm that returns $b \in \{0,1\}^n$ such that $s \cdot b = 0$ and $g \cdot b = 1$ if $g \neq s$. Moreover, the algorithm runs in linear time in n and in the time required to compute f. Especially, when f can be computed in polynomial time of size n, the algorithm runs in polynomial time of size n.*

Proof. Let

$$g_1(a) = \max\{f(a), f(a \oplus g)\} \text{ and}$$

$$g_2(a) = \begin{cases} 0 & \text{if } f(a) > f(a \oplus g), \\ 1 & \text{otherwise.} \end{cases}$$

Then, we define two operations U_{g_1} and U_{g_2} by

$$|a\rangle|0\rangle \xrightarrow{U_{g_1}} |a\rangle|g_1(a)\rangle \text{ and}$$

$$|a\rangle \xrightarrow{U_{g_2}} (-1)^{g_2(a)}|a\rangle.$$

If the function f can be computed in $T_f(n)$, it is clear that each running time of U_{g_1} and U_{g_2} is $O(T_f(n))$ time.

Now, let us show a quantum algorithm to find b. First, we apply $W_{H_{2^n}}$ to the first register on $|0\rangle|0\rangle$ and then apply U_{g_1}, where we call s_1 (resp. s_2) for $|s_1\rangle|s_2\rangle$ the first (resp. the second) register.

$$\begin{aligned} |0\rangle|0\rangle &\xrightarrow{W_{H_{2^n}}} \frac{1}{2^{n/2}} \sum_{a\in\{0,1\}^n} |a\rangle|0\rangle \\ &\xrightarrow{U_{g_1}} \frac{1}{2^{n/2}} \sum_{a\in\{0,1\}^n} |a\rangle|g_1(a)\rangle \end{aligned}$$

Let $K \subseteq \{0,1\}^n$ be a subset such that for any $a \in \{0,1\}^n$,

$$|K \cap \{a, a\oplus s, a\oplus g, a\oplus g\oplus s\}| = 1.$$

Since $g_1(a) = g_1(a\oplus g)$ and $g_1(a) \neq g_1(a')$ for any $a \neq a' \in K$,

$$\frac{1}{2^{n/2}} \sum_{a\in\{0,1\}^n} |a\rangle|g_1(a)\rangle = \frac{1}{2^{n/2}} \sum_{a\in K} (|a\rangle + |a\oplus s\rangle + |a\oplus g\rangle + |a\oplus g\oplus s\rangle)|g_1(a)\rangle.$$

Next, we apply U_{g_2} to the first register. Since $(-1)^{g_2(a)} = -(-1)^{g_2(a\oplus g)}$ and $(-1)^{g_2(a)} = (-1)^{g_2(a\oplus s)}$,

$$\begin{aligned} &\frac{1}{2^{n/2}} \sum_{a\in K} (|a\rangle + |a\oplus s\rangle + |a\oplus g\rangle + |a\oplus g\oplus s\rangle)|g_1(a)\rangle \\ &\quad \xrightarrow{U_{g_2}} \frac{1}{2^{n/2}} \sum_{a\in K} (-1)^{g_2(a)}(|a\rangle + |a\oplus s\rangle - |a\oplus g\rangle - |a\oplus g\oplus s\rangle)|g_1(a)\rangle. \end{aligned}$$

Finally, we apply $W_{H_{2^n}}$ to the first register. From $(\alpha\oplus\beta)\cdot\gamma = (\alpha\cdot\gamma)\oplus(\beta\cdot\gamma)$, we have $(-1)^{(\alpha\oplus\beta)\cdot\gamma} = (-1)^{\alpha\cdot\gamma}(-1)^{\beta\cdot\gamma}$. Thus,

$$\begin{aligned} &\frac{1}{2^{n/2}} \sum_{a\in K} (-1)^{g_2(a)}(|a\rangle + |a\oplus s\rangle - |a\oplus g\rangle - |a\oplus g\oplus s\rangle)|g_1(a)\rangle \\ &\xrightarrow{W_{H_{2^n}}} \frac{1}{2^n} \sum_{a\in K} \sum_{b\in\{0,1\}^n} (-1)^{g_2(a)} \Big((-1)^{a\cdot b} + (-1)^{(a\oplus s)\cdot b} \\ &\qquad\qquad - (-1)^{(a\oplus g)\cdot b} - (-1)^{(a\oplus g\oplus s)\cdot b}\Big)|b\rangle|g_1(a)\rangle \\ &\quad = \frac{1}{2^n} \sum_{a\in K} \sum_{b\in\{0,1\}^n} (-1)^{g_2(a)+a\cdot b}(1 + (-1)^{s\cdot b})(1 - (-1)^{g\cdot b})|b\rangle|g_1(a)\rangle. \end{aligned}$$

Then, by measuring the first register, we can obtain b which satisfies $s \cdot b = 0$ and $g \cdot b = 1$. Moreover, total running time of this quantum algorithm is $O(n + T_f(n))$ time. □

A set $B \subseteq \{0,1\}^n$ is defined *linearly independent* if $b \neq \bigoplus_{b' \in B'} b'$ for every $b \in B$ and for any subset $B' \subseteq B - \{b\}$.

Lemma 2. *Given a linearly independent set $B \subseteq \{0,1\}^n$, there exists a polynomial time algorithm that returns a nonzero $g \in \{0,1\}^n$ such that $g \cdot b = 0$ for every $b \in B$. Moreover, g is unique if $|B| = n - 1$.*

Proof. A nonzero $g \in \{0,1\}^n$ which satisfies the condition of this lemma can be find by using a standard technique in the coding theory (e.g., see [9]). Suppose C is a binary linear code such that B is a basis of it. Then, the dual code of C, $C^{\perp}$, can be defined as $C^{\perp} = \{c \in \{0,1\}^n \mid c \cdot b = 0 \text{ for every } b \in B\}$. From [9], if B is given, a basis G of $C^{\perp}$ can be obtained in polynomial time in $|B|$ and n. Since $|B| < n$, G can be obtained in polynomial time in n. Then, since $G \subseteq C^{\perp}$, every $g \in G$ satisfies the condition of this lemma.

Furthermore, it is known that $|G| = n - |B|$, and $C^{\perp} = \{0, g\}$ if $G = \{g\}$ for some nonzero g [9]. Therefore, the nonzero g which satisfies the condition of this lemma is unique if $|B| = n - 1$. □

This lemma implies that we can certainly find a nonzero s if there exists a linearly independent set $B \subseteq \{0,1\}^n$ such that $|B| = n - 1$ and $s \cdot b = 0$ for every $b \in B$. The problem is how to obtain such a linearly independent set. The following lemma show that we can obtain such a linearly independent set by selecting a good g for each application of Lemma 1.

Lemma 3. *Let B ($|B| < n - 1$) be a linearly independent set, and $g \in \{0,1\}^n$ an arbitrary nonzero element such that $g \cdot b = 0$ for every $b \in B$. Then for any b' such that $g \cdot b' = 1$, $B \cup \{b'\}$ is also linearly independent.*

Proof. Since $g \cdot (b \oplus b') = (g \cdot b) \oplus (g \cdot b')$, we have $g \cdot (\bigoplus_{b \in B'} b) = 0$ for any subset $B' \subseteq B$. Thus, $b' \neq (\bigoplus_{b \in B'} b)$ for any b' such that $g \cdot b' = 1$ and for any subset $B' \subseteq B$. Therefore, $B \cup \{b'\}$ is linearly independent. □

By applying Lemma 1 and Lemma 2 as subroutines, the following theorem is obtained.

Theorem 1. *Let f be a function that satisfies the condition of Simon's problem. Then, there exists a quantum algorithm that solves Simon's problem in polynomial time in n and in the time required to compute f in worst case. Especially, when f can be computed in polynomial time of size n, the algorithm runs in polynomial time of size n in worst case.*

Proof. First, we construct a quantum algorithm for solving Simon's problem.

– Step 1: Set $B := \emptyset$.
– Step 2: Select a nonzero $g \in \{0,1\}^n$.

– Step 3: Repeat the following steps while $g \neq s$.
  - Step 3-1: Find a nonzero $b \in \{0,1\}^n$ such that $s \cdot b = 0$ and $g \cdot b = 1$ by applying Lemma 1 for g, and set $B := B \cup \{b\}$.
  - Step 3-2: Find a nonzero g' such that $g' \cdot b = 0$ for every $b \in B$ by applying Lemma 2, and set $g := g'$.

– Step 4: Return g (i.e., s).

Now, we show the correctness of our algorithm. By Lemma 3, Therefore, when we find the set B of size $|B| = n - 1$, we can find the nonzero s because $s \cdot b = 0$ for every $b \in B$

Finally, let us consider the running time of our algorithm. Let $T_f(n)$ be the time required to compute f, and $G(n)$ the time required by applying Lemma 2. Note that $G(n)$ is some polynomial of n. The main step is Step 3 (other steps are at most $O(n)$ time). The running time of Step 3-1 is $O(n + T_f(n))$ time by Lemma 1 and the running time of Step 3-2 is $O(G(n))$ time. Step 3 is repeated at most $n - 1$ times. Then, the total running time of our algorithm is $O(n^2 + nT_f(n) + nG(n))$ time in worst case. □

Note that the term $nG(n)$ in the running time can be improved, because the running time of an application of Lemma 2 depends not only on n but also on the size of B.

4 Conclusions

We showed a quantum algorithm that solves Simon's problem not in expected sense but in worst case. This result is not new because another algorithm to solve the problem in worst case [3]. However, our algorithm is much simpler than their algorithm. Moreover, it implies by Lemma 1 that an extended version of the problem in [3] (i.e., the range of the function f is extended from $\{0,1\}^{n-1}$ to $\{0,1\}^m$ for any $m \geq n$) can also be efficiently solved.

Furthermore, by a simple modification, our algorithm can efficiently solve the extended Simon's problem defined in [4].

Extended Simon's problem [4]

> *Let $f : \{0,1\}^n \to \{0,1\}^m$ with $n \leq m$ be a function such that for any $x \neq x' \in \{0,1\}^n$ and a set $S \subseteq \{0,1\}^n$,*
>
> $$f(x) = f(x') \quad \textit{iff} \quad x' = x \oplus s \quad \textit{for some } s \in S.$$
>
> *Then, find a basis of S.*

This problem can be given in such a way that the size of S is unknown.

Shor showed that there exists polynomial time quantum algorithms for factoring and finding discrete logarithm, but his algorithm is also constructed in expected sense[10,11]. Therefore, in this related work, the most interesting open problem is whether there exist a polynomial time quantum algorithm for factoring or finding discrete logarithm in worst case. Our result may help to solve these open problems.

References

1. Bernstein, E., Vazirani, U. V.: Quantum complexity theory (Extended Abstract). In Proc. of the 25th ACM Symp. on Theory of Computing (1993) 11–20 229, 230
2. Bernstein, E., Vazirani, U. V.: Quantum complexity theory. SIAM J. Comput. **26** (1997) 1411–1473 229, 230
3. Brassard, G., Høyer, P.: On the power of exact quantum polynomial time. Manuscript (available as quant-ph/9612017 at http://xxx.lanl.gov) (1996) 235
4. Brassard, G. , Høyer, P.: An exact quantum polynomial-time algorithm for Simon's problem. In Proc. Fifth Israeli Symposium on Theory of Computing and Systems (June 1997) 12–23 230, 235
5. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. In Proc. R. Soc. Lond. **A400** (1985) 97–117 229, 230
6. Deutsch, D. , Jozsa, R.: Rapid solution of problems by quantum computation. In Proc. R. Soc. Lond. **A 439** (1992) 553–558 229
7. Feynman, R. P.: Simulating Physics with Computers. Int. J. Theor. Phys. **21** (1982) 467–488 229
8. Grover, L. K.: A fast quantum mechanical algorithm for database search. In Proc. of the 28th ACM Symp. on Theory of Computing (1996) 212–219 230
9. van Lint, J.H.: Introduction to Coding Theory (Second Edition). Springer-Verlag (1991) 234
10. Shor, P. W.: Algorithms for quantum computation: discrete logarithms and factoring. In Proc. of the 35th Ann. Symp. on Foundations of Computer Science (1994) 124–134 235
11. Shor, P. W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26** (1997) 1484–1509 235
12. Simon, D. R.: On the power of quantum computation. In Proc. of the 35th Ann. Symp. on Foundations of Computer Science (1994) 116–123 229, 231
13. Simon, D. R.: On the power of quantum computation. SIAM J. Comput. **26** (1997) 1474–1483 229, 231

Generalized Graph Colorability and Compressibility of Boolean Formulae

Richard Nock, Pascal Jappy, and Jean Sallantin

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
161, rue Ada, 34392 Montpellier, France
{nock,jappy,js}@lirmm.fr

Abstract. In this paper, we study the possibility of Occam's razors for a widely studied class of Boolean Formulae : Disjunctive Normal Forms (DNF). An Occam's razor is an algorithm which compresses the knowledge of observations (examples) in small formulae. We prove that approximating the minimally consistent DNF formula, and a generalization of graph colorability, is very hard. Our proof technique is such that the stronger the complexity hypothesis used, the larger the inapproximability ratio obtained. Our ratio is among the first to integrate the three parameters of Occam's razors : the number of examples, the number of description attributes and the size of the target formula labelling the examples. Theoretically speaking, our result rules out the existence of efficient deterministic Occam's razor algorithms for DNF. Practically speaking, it puts a large worst-case lower bound on the formulae's sizes found by learning systems proceeding by rule searching.

1 Introduction

The learnability of Disjunctive Normal Form formulae (disjunctions of conjunctions) is a central problem in machine learning [27]. In 1984, Valiant studies the learnability of this class, and remarks that "*the attraction of this class is that humans appear to like it for representing knowledge as is evidenced, for example, by the success of the production system paradigm and of Horn clause logics*" [35]. He proves that a subclass of DNF is learnable, and leaves as an open problem whether the whole class is learnable. Since then, many theoretical studies have investigated the learnability of DNF or subclasses [19,27,2,1,4,5,8,15,20,23,28].

Simultaneously, many programs for machine learning were designed to learn efficiently from examples. In each of these programs, the algorithms has access to a learning sample and tries to build a small function approximating as best as possible the observed examples. According to [10], systems that learn sets of rules (DNF in the Boolean framework) have a number of desirable properties : they are easy to understand, they can outperform decision-tree learning algorithms on many problems, they have a natural and familiar first-order version (Prolog predicates), and techniques for learning propositional rule sets can often be extended to the first-order case [29]. Many learning algorithms either build rules, or have a stage consisting in searching for rules that are postprocessed, or are aimed at producing formulae that can be easily translated into rules, [7,10,9,12,21,24,25,26,31,30,29,32,33,36] and many others.

Practical and theoretical results often focus on an aspect of approximation they both share : given a set of examples (*e.g.* description of animals for which we know if they have or not some illness), can we devise an efficient algorithm which can find a small formula (*i.e.* a mean of classifying observations) consistent with all examples (making no errors)?
Theoretically speaking, this aspect is often related to the principle of Occam's razors [6]: an Occam's razor for a class (*i.e.* a set) of concept representations C (each of which is a function mapping observations to classes) is an algorithm that, given a learning sample LS whose labels are given by some unknown target concept $t \in C$, can produce in time polynomial in $|LS|$, n (the number of description variables), $|t|$ a formula $h \in C$ satisfying to the two following conditions : h is consistent with LS (it does not make errors) and has size satisfying $|h| \leq |LS|^a(n|t|)^b$ (with $0 \leq a < 1$ and $b \geq 0$). The principle of Occam's razors [18] states that in order to learn, a system should compress the information contained in the examples. This principle was originally stated by philosoph William of Occam (1285-1349), and led to theoretical results in the PAC-learning model of Valiant [34] : learning is in fact equivalent to finding Occam's razors [18].
Practically speaking, machine learning algorithms producing rules are almost always aimed at producing small sets of rules, because they are easy to understand for the non-expert, and they appear to be sufficient on many problems [17].

The principal result on the inapproximability of DNF comes from [19]. They show that DNF is as hard to approximate as a proglem related to a generalization of Graph Colorability (which does not have commonpoints with ours). [13] prove that Graph Colorability is hard to approximate to within n^δ ($\forall 0 < \delta < 1$). Using the reduction of [19], we are able to show that the consistent DNF with minimal size is not approximable (in size) to within n^δ ($\forall 0 < \delta < 1$), where n is the number of description variables of the examples, a measure of the problem's complexity. In this paper, we first prove that the upperbound of δ can be removed : the result holds in fact $\forall \delta > 0$. We go further into negative results, and prove that size-$|t|$ DNF cannot be approximated by DNF having size not greater than $|LS|^a n^b |t|^c$. a, b, c are any constants satisfying $\frac{1}{19} > a \geq 0$, $b \geq 0$, and $1 + \frac{1}{145} > c \geq 0$. $|t|$ is the number of monomials (conjunctions) of the target concept. This proof is stated under the hypothesis $NP \not\subset ZPP$, where ZPPdenotes the class of langages decidable by a random expected polynomial-time algorithm that makes no errors [3].

In order to achieve our result, we firstly prove an equivalence of approximating DNF with a generalization of graph colorability. We then prove our result on the inapproximability of DNF by proving an inapproximability result on the generalization of Graph Colorability.

2 Equivalence between Approximating DNF and a Generalization of Graph Colorability

In this paper, we are interested in approximating optimization problems. An optimization problem contains an instance, the definition of a feasible solution, and a cost function defined for any feasible solution. The aim of any approximation algorithm is to find feasible solutions whose cost (*e.g.* number of colors for coloring a graph) is as close as possible from the problem's optimum. We also use the cost notion for instances : the cost of an instance is the optimal cost among all feasible solutions for this instance.

Let $\mathcal{F}$ be a class of Boolean formulae; any of its elements, f, is a function $f : \{0,1\}^n \rightarrow \{0,1\}$. An element $x \in \{0,1\}^n$ is an *example*. It is composed of n binary *variables* $\{x_1, ..., x_n\}$ assigned in $\{0,1\}$ (the negative and positive *literals*). The value $f(x)$ is the *class* $\in \{0,1\}$ that f assigns to x (the *negative* and *positive* class). The size of any formula f is denoted $|f|$. We investigate the class of DNF, set of formulae described as a disjunction of monomials. A monomial is a conjunction ($\wedge$) of literals (a literal is a Boolean descriptor, taking value either True or False). We are interested by the possibility, for some efficient (Ptime) algorithm, to approximate the following minimization problem:

- **Name** : *Opt*(DNF)
- **Instance** : A learning sample *LS*
- **Feasible Solutions** : Formulae from DNF consistent with *LS*
- **Cost Function** : Size of the formula (number of monomials)

It is well-known [19] that this problem is as hard as the problem *Opt* (Independant-set cover) (this is the same as the graph-coloring problem [14]; however, this name is convenient for our proofs). We show in this paper that it is in fact as hard as a generalization of the *Opt*(Independant-set cover) problem (for any integer $k > 0$, $[k]$ denotes the set $\{1, 2, ..., k\}$):

Definition 1 ***Opt*(Multi independent-set cover)**

- **Name** *: Opt(Multi independent-set cover)*
- **Instance** *:* $G^{\oplus} = (X^{\oplus}, E^{\oplus})$*, a graph presenting the following form : for some positive integer* d*,* $X^{\oplus}$ *is partitionned into* $X_1, ..., X_d$ *such that if* $d > 1$*,* $\forall 1 \leq i < j \leq d$*, none of the edges between* X_i *and* X_j *are in* $E^{\oplus}$*.*
- **Feasible Solutions** *: a cover of* $X^{\oplus}$ *in subsets* $s_1, ..., s_k$ *such that :*
  1. $\forall 1 \leq i \leq k, \forall 1 \leq j \leq d$*,* $s_i \cap X_j \neq \emptyset$ *and induces an independent set in* X_j*.*
  2. $\forall (x_1, x_2, ..., x_d) \in \prod_{i=1}^{d} X_i, \exists j \in [k] : \forall l \in [d], x_l \in s_j$
- **Cost Function** *:* k*, that is, the number of subsets used to cover* $X^{\oplus}$*.*

Note that every graph can be represented according to the preceeding definition, but the uniqueness of the representation is not ensured for a given graph. The cost of a graph G instance of *Opt*(Independant-set cover) is usually written $\chi(G)$. We note $\chi_g(G^{\oplus})$ as the cost of a graph $G^{\oplus}$ instance of *Opt*(Multi independent-set cover). The following proposition states the equivalence between these two problems (proof omitted due to space limitations).

Proposition 1 Equivalence of approximating *Opt*(Multi independent-set cover) and *Opt*(DNF) : *For any graph $G^{\oplus} = (X^{\oplus}, E^{\oplus})$ instance of Opt(Multi independent-set cover), we can create in time polynomial in $|X^{\oplus}|$ a set of examples LS such that if there exists a feasible solution to Opt(Multi independent-set cover) whose cost is k, then we can create in Ptime a DNF having no more than k monomials and consistent with LS. Reciprocally, if there exists a DNF of size k consistent with LS, then (i) we can suppose without loss of generality that it is monotonous (no negative literals), and (ii) we can generate in Ptime a feasible solution to Opt(Multi independent-set cover) whose cost does not exceed k.*

3 *Opt*(Multi Independent-Set Cover) is Hard to Approximate

This section is devoted to the proof of the following theorem.

Theorem 1 *Unless NP⊆ZPP, Opt(Multi independent-set cover) is not approximable to within*

$$\rho^{\oplus} = \left(\max_{1\leq i\leq n}\{|E_i| + |V_i|\}\right)^{da} \left(d \max_{1\leq i\leq n}\{|V_i|\}\right)^{b} (\chi_g(G^{\oplus}))^{c-1}$$

where $G_i = (X_i, E_i)$ is the subgraph of $G^{\oplus}$ induced by X_i. The result holds

- *$\forall b \geq 0$,*
- *$\forall \frac{1}{19} > a \geq 0$,*
- *$\forall 1 + \frac{1}{145} > c \geq 0$.*

This theorem means that no polynomial-time (Ptime) algorithm can guarantee to find, from an instance $G^{\oplus}$ of *Opt*(Multi independent-set cover), a solution whose cost does not exceed $\chi_g(G^{\oplus}) \times \rho^{\oplus}$. The proof technique basically relies on multiplying d instances (or stairs) of *Opt*(Independant-set cover) to form an instance of *Opt*(Multi independent-set cover), without linking each stair (graphs instance of *Opt*(Independant-set cover)) to the others. Ideally, we would like to obtain a relationship such as

$$\chi_g(G^{\oplus}) = \chi(G)^d \tag{1}$$

which would ease a lot the proof since it would also blow up any inapproximability ratio ρ for *Opt*(Independant-set cover) to $\rho^{\oplus} = \rho^d$ for *Opt*(Multi independent-set cover). However, this relationship is *not* true for any graph instance of

Opt(Independant-set cover), that is why we need to build particular, very hard to solve instances of *Opt*(Independant-set cover). Figure 1 presents the building of a hard instance G of *Opt*(Independant-set cover), from two reductions τ_1 and τ_2. Reduction τ_3^d is the stacking-up of d instances of *Opt*(Independant-set cover) to form an instance of *Opt*(Multi independent-set cover). The main theorem we

$$\begin{tikzcd}
 & & & \text{Instance of } Opt\text{(Independant set cover)} \arrow[d, dashed] & \\
\text{“SAT”} \arrow[r, dotted] & G'' \arrow[r, "\tau_1"] & G' \arrow[r, "\tau_2"] & G \arrow[r, "\tau_3^d"] & G^{\oplus} \\
 & \text{Instance of } Opt\text{(Independant set)} \arrow[u, dashed] & & &
\end{tikzcd}$$

Fig. 1. Scheme of the reduction $\tau_1 \circ \tau_2 \circ \tau_3^d$.

need in this part is the following:

Theorem 2 *(From [16]) Unless $NP \subseteq ZPP$, $\forall 0 \leq \delta < 1$, Opt(Independent set) is not approximable to within $\rho'' = n(G'')^{\delta}$.*

We suppose for the sake of simplicity, as it is pointed out in [13,16], that theorem 2 is proven from the decision problem “SAT”. It means that there exists a reduction from “SAT” to *Opt*(Independent set) such that

- Any graph G'' obtained from a satisfiable instance of “SAT” satisfies $\alpha(G'') = g$.
- Any graph G'' obtained from an unsatisfiable instance of “SAT” satisfies $\alpha(G'') < g/n(G'')^{\delta}$, $\forall 0 < \delta \leq 1$.

We now describe the two first reductions, τ_1 and τ_2. Define as K_r the complete graph over r vertices.

Definition 2 τ_1 **[22]** *From G'', we create the graph product $G' = K_r \times G''$ with $r \geq \alpha(G'')$ as follows. Each vertex of G' is a pair $< i, v >$ where $i = 1, ..., r$ is a vertex from K_r and v is a vertex from G''. Sets of vertices with the same first component induce a clique in G'. Two vertices $< i, v >, < j, w >$ from different cliques are adjacent iff their second components v and w are equal or adjacent vertices in G''.*

(In that follows, we fix $r = n(G'')$). As pointed out in [22], Section 2.1, we have $\alpha(G'') = \alpha(G')$. Let Z_p be the field of integers modulo p.

Definition 3 τ_2 **[22]** *Let G' be a graph whose vertices are partitionned into cliques $C_1, ..., C_r$. Let p be a prime at least as large as*

$$\max\{\max_i |C_i|, r/\alpha(G'), \sqrt{r}\}$$

The image of G' by τ_2, G, has vertices described as quadruplets $< i, k, y, w >$, where $i = 1, ..., r$, $k \in Z_p$, $y \in Z_p^2$, and $w \in Z_p^2$. Let $u_A =< i_A, k_A, y_A, w_A >$ and $u_B =< i_B, k_B, y_B, w_B >$ be two vertices of G. The two vertices are **not** *adjacents iff $(\exists s \in Z_p)(\exists z \in Z_p^2)$ such that:*

- *$< i_A, k_A - s >\in G'$, $< i_B, k_B - s >\in G'$ and they are not adjacent.*
- *$w_A = (k_A - s)y_A + z$ and $w_B = (k_B - s)y_B + z$.*

Like [22], we apply transformations τ_1 and τ_2 to any graph G'' instance of *Opt*(Independent set). We have

Proposition 2 *G satisfies:*

1. *$n(G) = rp^5$ ([22], part 2.1)*
2. *$\alpha(G) = p^2\alpha(G')$ ([22], corollary 2.2)*
3. *If G is built from a satisfiable instance of "SAT", then $\chi(G) = n(G)/\alpha(G)$ ([22], theorem 2.3)*
4. *$p^2\chi(G) \leq n(G)$*

(proof of [4] omitted due to space limitations). The third reduction, τ_3^d, with $d > 0$ an integer, simply consists in stacking-up d times G without linking each "stair" to the others. Let $G^\oplus = \tau_3^d(G)$. Since there are $n(G)^d$ d-tuples containing one vertex from each $X_1, ..., X_d$, since any set from any solution to *Opt*(Multi independent-set cover) contains at most $\alpha(G)^d$ of these d-tuples, and since any of these d-tuples are covered, it comes $\chi_g(G^\oplus) \geq \frac{n(G)^d}{\alpha(G)^d}$. Furthermore, making the d-times cross-product of the independent sets of a solution to *Opt*(Graph colorability) lead to a feasible solution to *Opt*(Multi independent-set cover) whose cost satisfies $\chi_g(G^\oplus) \leq \chi(G)^d$. Consequently,

$$\frac{n(G)^d}{\alpha(G)^d} \leq \chi_g(G^\oplus) \leq (\chi(G))^d \tag{2}$$

We refine these inequations. Any graph $G^\oplus$ corresponding to a satisfiable instance of "SAT" leads by proposition 2 to $\chi(G) = n(G)/\alpha(G)$. Therefore, $\chi_g(G^\oplus) = \chi(G)^d$. Any graph $G^\oplus$ corresponding to an unsatisfiable instance "SAT" leads by theorem 2 and proposition 2 to $\forall 0 \leq \delta < 1, \chi_g(G^\oplus) > \left(p^3 n(G'')^\delta\right)^d$. From proposition 2, we also get $\forall 0 \leq \delta < 1, p^3 n(G'')^\delta \geq \frac{\chi(G)p^5 n(G'')^\delta}{n(G)}$. But $n(G)/p^5 = r = n(G'')$. Therefore $\forall 0 \leq \delta < 1, p^3 n(G'')^\delta \geq \frac{\chi(G)}{n(G'')^{1-\delta}}$. Therefore, for any graph $G^\oplus$ corresponding to instances of "SAT" either satisfiable or not, we have:

$$\forall 0 \leq \delta < 1, \left(\frac{\chi(G)}{n(G'')^{1-\delta}}\right)^d \leq \chi_g(G^\oplus) \leq (\chi(G))^d \tag{3}$$

This relationship is central for our proof; although it is much weaker than equation 1, it is still sufficient to prove theorem 1. However, there are two problems left : how can we use 3 to prove $\rho^\oplus$, and can we choose d constant, so that τ_3^d is

Ptime ? We first solve the first problem. From any “SAT” instance (theorem 2) transformed in a graph G using reductions τ_1 and τ_2, depending on whether it is satisfiable or not, there exists $g > 0$ (it is a function of the “SAT” instance, [22]) such that either $\chi(G) = p^3$, or $\chi(G) > p^3 n(G'')^\delta$, $\forall 0 < \delta \leq 1$.
For any satisfiable instance of “SAT”, we get from inequations 3 : $\chi_g(G^\oplus) \leq \chi(G)^d = (p^3)^d$. For any unsatisfiable instance of “SAT”, we get from inequations 3 : $\chi_g(G^\oplus) \geq \left(\frac{\chi(G)}{n(G'')^{1-\delta}}\right)^d \geq (p^3)^d \times (n(G'')^{2\delta-1})^d$. In order to prove theorem 1, it is sufficient to show a “gap” greater than $\rho^\oplus$ between $(p^3)^d$ and $(p^3)^d \times (n(G'')^{2\delta-1})^d$. That is, we need to find d such that

$$(n(G'')^{2\delta-1})^d > \rho^\oplus \Rightarrow d > \frac{\log \rho^\oplus}{\log(n(G'')^{2\delta-1})} \qquad (4)$$

under the constraint $\delta > 1/2$. And we need to show that d is constant.

We now check the constant value for d satisfying inequation 4. [22] choose for p a prime at least as large as $\max\left\{\max_i |C_i|, \frac{r}{\alpha(G'')}, \sqrt{r}\right\}$, which is $n(G'')$ in our case. We also have

Proposition 3 [11] *There exists a constant* $0 < \alpha < 1$ *such that for any positive integer* M*, there exists a prime falling in the interval* $[M; M^{1+\alpha}]$*. Furthermore, we can fix* $\alpha = 11/20$*.*

In our case, we can therefore suppose that $n(G'') \leq p \leq n(G'')^{1+\alpha}$. We fix $\frac{49}{50} < \delta < 1$ if $c \leq 1$, and $\frac{49}{50} + \frac{113(c-1)}{40} < \delta < 1$ otherwise (remark that $\frac{1}{2} < \delta < 1$). Fix

$$d = \lceil \frac{f(a,b,c,\alpha,\delta) + (6+5\alpha)b}{2\delta - 1} \rceil \qquad (5)$$

with

$$f(a,b,c,\alpha,\delta) = 2 \times \left[\frac{1}{a(12+10\alpha)+\max\{0;(c-1)(4+3\alpha)\}} - \frac{1}{2\delta-1}\right]^{-1} \times \left[\frac{(6+5\alpha)b}{2\delta-1} + 1\right]$$

Arithmetic calculation gives

Fact 1 *The choice of* δ *leads that* $f(a,b,c,\alpha,\delta)$ *is positive. Furthermore,*

$$f(a,b,c,\alpha,\delta) - d\,(a(12+10\alpha) + \max\{(c-1)(4+3\alpha); 0\}) > 0$$

Fact 1 leads to $d^b < n(G'')^{f(a,b,c,\alpha,\delta) - d(a(12+10\alpha)+(c-1)(4+3\alpha))}$, at least for sufficient large-sized graphs. Proposition 2 leads to

$$\chi_g(G^\oplus) \leq \chi(G)^d \leq \left(\frac{n(G)}{p^2}\right)^d = (rp^3)^d \leq n(G'')^{d(4+3\alpha)} \qquad (6)$$

The fact that vertices of G can be partitionned into independent sets of size p^2 (proposition 2) leads to (for convenience, we fix $e(G)$ and $n(g)$ to be respectively the number of edges and the number of vertices of G) : $e(G) \leq p^4 \frac{(rp^3)!}{(rp^3-2)!2!} = \frac{rp^7(rp^3-1)}{2}$. From this and the fact that $n(G) = rp^5 \leq n(G'')^{6+5\alpha}$, we get $e(G) + n(G) \leq \frac{rp^7(rp^3-1)}{2} + rp^5$, and therefore $e(G) + n(G) < r^2p^{10} = n(G'')^{12+10\alpha}$. Putting it altogether, we get that

$$\rho^{\oplus} = \left(\max_{1\leq i\leq n}\{|E_i| + |V_i|\}\right)^{da} \left(d \max_{1\leq i\leq n}\{|V_i|\}\right)^{b} (\chi_g(G^{\oplus}))^{c-1}$$

implies whenever $c \geq 1$

$$\begin{aligned}\rho^{\oplus} &< n(G'')^{(12+10\alpha)ad} \times n(G'')^{f(a,b,c,\alpha,\delta)-(12+10\alpha)ad-d(c-1)(4+3\alpha)} \\ &\quad \times n(G'')^{(6+5\alpha)b} \times n(G'')^{d(c-1)(4+3\alpha)}\end{aligned}$$

and therefore $\rho^{\oplus} < n(G'')^{f(a,b,c,\alpha,\delta)+(6+5\alpha)b}$. When $c < 1$, similar calculation leads again to $\rho^{\oplus} < n(G'')^{f(a,b,c,\alpha,\delta)+(6+5\alpha)b}$. We get

$$\frac{\log \rho^{\oplus}}{\log(n(G'')^{2\delta-1})} < \frac{f(a,b,c,\alpha,\delta) + (6+5\alpha)b}{2\delta - 1} \tag{7}$$

But the choice of d also gives

$$\frac{f(a,b,c,\alpha,\delta) + (6+5\alpha)b}{2\delta - 1} \leq d \tag{8}$$

d is therefore constant, and satisfies inequation 4. The proof of theorem 1 is completed. From proposition 1, from the fact that $|LS| \leq (\max_{1\leq i\leq n}\{|E_i| + |V_i|\})^d$, $n \leq d \max_{1\leq i\leq n}\{|V_i|\}$, and the target concept's size satisfies $|t| = \chi_g(G^{\oplus})$, we obtain

Theorem 3 Non-approximability of DNF *Unless $NP \subseteq ZPP$, there cannot exist an Occam's razor for DNF finding formulae whose size does not exceed $|LS|^a n^b |t|^c$, where $|t|$ is the size of the target concept, and $|LS|$ the size of the learning sample. The result is true even if we suppose that the target concept belongs to monotone-DNF. a, b, c are any constants satisfying:*

$$\frac{1}{19} > a \geq 0 \;\;;\;\; b \geq 0 \;\;;\;\; 1 + \frac{1}{145} > c \geq 0$$

4 Conclusion

Recall that an Occam's razor for a class of Boolean formulae C is an algorithm that, given a learning sample LS whose labels are given by some unknown target concept $t \in C$, can produce in time polynomial in $|LS|$, n, $|t|$ a formula $h \in C$ satisfying to the two following conditions : h is consistent with LS and $|h| \leq |LS|^a (n|t|)^b$, with $a, b > 0$ and $a < 1$. With our reasonable complexity hypothesis, our result does not rule out any possibility of deterministic Occam's razors, even if efficient Occam's razors are proven impossible. However, the advantage of our reduction technique is that, the higher the time alloted for τ_3^d (thus, the higher d), the higher the non-approximability ratio. [27] cite the complexity hypothesis $NP \not\subset \text{DTIME}(poly(2^{n^\epsilon}))$ (for some $\epsilon > 0$). With such hypotheses, adaptated to handle zero-error, probabilistic algorithms, it would be interesting to see how close to the Occam requirements the non-approximability ratio would come.

References

1. H. Aizenstein and L. Pitt. Exact learning of read-k-disjoint DNF and not-so-disjoint-DNF. In *Proc. of the 5 th International Conference on Computational Learning Theory*, pages 71–76, 1992. 237
2. H. Aizenstein and L. Pitt. On the learnability of Disjunctive Normal Form formulas. *Machine Learning*, 19:183–208, 1995. 237
3. J. L. Balcazar, J. Diaz, and J. Gabarro. *Structural Complexity I.* Springer Verlag, 1988. 238
4. U. Berggren. Linear time deterministic learning of k-term-DNF. In *Proc. of the 6 th International Conference on Computational Learning Theory*, pages 37–40, 1993. 237
5. A. Blum, R. Khardon, E. Kushilevitz, L. Pitt, and D. Roth. On learning read-k-satisfy-j DNF. In *Proc. of the 7 th International Conference on Computational Learning Theory*, pages 110–117, 1994. 237
6. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, pages 377–380, 1987. 238
7. C. Brunk and M. Pazzani. Noise-tolerant relational concept learning. In *Proc. of the 8 th International Conference on Machine Learning*, 1991. 238
8. N. H. Bshouty, Z. Chen, S. E. Decatur, and S. Homer. On the learnability of z_n-DNF formulas. In *Proc. of the 8 th International Conference on Computational Learning Theory*, pages 198–205, 1995. 237
9. W. W. Cohen. PAC-learning a restricted class of recursive logic programs. In *Proc. of AAAI-93*, pages 86–92, 1993. 238
10. W. W. Cohen. Fast effective rule induction. In *Proc. of the 12 th International Conference on Machine Learning*, pages 115–123, 1995. 237, 238
11. C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, pages 1–14, 1997. 243
12. L. de Raedt. Iterative concept learning and construction by analogy. *Machine Learning*, pages 107–150, 1992. 238
13. U. Feige and J. Kilian. Zero knowledge and the chromatic number. draft, 1996. 238, 241
14. M.R. Garey and D.S. Johnson. *Computers and Intractability, a guide to the theory of NP-Completeness.* Bell Telephone Laboratories, 1979. 239
15. S. A. Goldman and H. D. Mathias. Learning k-term-DNF formulas with an incomplete membership oracle. In *Proc. of the 5 th International Conference on Computational Learning Theory*, pages 77–84, 1992. 237
16. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *FOCS'96*, pages 627–636, 1996. 241

17. R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, pages 63–91, 1993. 238
18. M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* M.I.T. Press, 1994. 238
19. M.J. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. *Proceedings of the Nineteenth Annual A.C.M. Symposium on Theory of Computing*, pages 285–295, 1987. 237, 238, 239
20. R. Khardon. On using the fourier transform to learn disjoint DNF. *Information Processing Letters*, pages 219–222, 1994. 237
21. N. Lavrac, S. Dzeroski, and M. Grobelnik. Learning non-recursive definitions of relations with linus. In *European Working Session in Learning*, 1991. 238
22. K. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. of the 25 th Symposium on the Theory of Computing*, pages 286–293, 1993. 241, 242, 243
23. Y. Mansour. An $O(n^{log\ log\ n})$ algorithm for dnf under the uniform distribution. In *Proc. of the 5 th International Conference on Computational Learning Theory*, pages 53–61, 1992. 237
24. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Inductive Logic Programming*, 1994. 238
25. R. Nock and O. Gascuel. On learning decision committees. In *Proc. of the 12 th International Conference on Machine Learning*, pages 413–420, 1995. 238
26. J. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 1990. 238
27. K. Pillaipakkamnatt and V. Raghavan. On the limits of proper learnability of subclasses of DNF formulae. In *Proc. of the 7 th International Conference on Computational Learning Theory*, pages 118–129, 1994. 237, 245
28. L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *J. ACM*, pages 965–984, 1988. 237
29. J. R. Quinlan. Learning logical definition from relations. *Machine Learning*, pages 239–270, 1990. 237, 238
30. J. R. Quinlan. *C4.5 : programs for machine learning.* Morgan Kaufmann, 1994. 238
31. J. R. Quinlan. MDL and categorical theories (continued). In *Proc. of the 12 th International Conference on Machine Learning*, pages 464–470, 1995. 238
32. C. Rouveirol. ITOU: induction of first-order theories. *Inductive Logic Programming*, 1992. 238
33. S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: a performance comparison of different lear ning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991. 238
34. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, pages 1134–1142, 1984. 238
35. L. G. Valiant. Learning disjunctions of conjunctions. In *Proc. of the 9 th IJCAI*, pages 560–566, 1985. 237
36. J Wnek and R. Michalski. Hypothesis-driven constructive induction in AQ17. In *Proc. of the 12 th IJCAI*, 1991. 238

On the Complexity of Free Monoid Morphisms

Klaus-Jörn Lange and Pierre McKenzie[⋆]

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
{lange,mckenzie}@informatik.uni-tuebingen.de

Abstract. We locate the complexities of evaluating, of inverting, and of testing membership in the image of, morphisms $h : \Sigma^* \to \Delta^*$. By and large, we show these problems complete for classes within NL. Then we develop new properties of finite codes and of finite sets of words, which yield image membership subproblems that are closely tied to the unambiguous space classes found between L and NL.

1 Introduction

Free monoid morphisms $h : \Sigma^* \to \Delta^*$, for finite alphabets Σ and Δ, are an important concept in the theory of formal languages (e.g. [6,11]), and they are relevant to complexity theory. Indeed, it is well known (e.g. [7]) that NP $=$ $\mathit{Closure}(\leq_m^{\mathrm{AC}^0}, \mathit{HOM}_{n.e.}) \subset \mathit{Closure}(\leq_m^{\mathrm{AC}^0}, \mathit{HOM}) = \mathit{R.E.}$, where $\leq_m^{\mathrm{AC}^0}$ denotes many-one AC^0-reducibility, HOM (resp. $\mathit{HOM}_{n.e.}$) is the set of morphisms (resp. nonerasing morphisms), and *Closure* denotes the smallest class of languages containing a finite nontrivial language and closed under the relations specified. Morphisms and their inverses also play a role in studying regular languages and "small" complexity classes: regular language varieties are closed under inverse morphisms [6], and the replacement of morphisms by "polynomial length M-programs", in the definition of recognition by a finite monoid M, allows automata to capture many subclasses of NC^1 [3,5,13].

Here we consider the complexity of evaluating inverse morphisms and morphisms $h : \Sigma^* \to \Delta^*$. Specifically, we consider the simple problem **eval** of computing the image of a word v under h, the problem **range** of determining whether a word $w \in h(\Sigma^*)$, and the problem **inv** of computing an element of $h^{-1}(w)$ given $w \in h(\Sigma^*)$. We examine the *fixed* setting, in which the morphism is input-independent, and the *variable* setting, in which the morphism is defined as part of the input.

The general framework of our results is summarized in the following figure. In the fixed case, the **eval** problem characterizes the relation between the classes NC^0, AC^0 and TC^0, and the problems **range** and **inv** are closely related to the class NC^1. Membership of **inv** in NC^1 is then to be contrasted with Håstad's result that there exists a fixed NC^0-computable function whose associated inversion problem is P-complete [10]. We also observe that $\mathit{Closure}(\leq_T^{\mathrm{AC}^0}, \mathit{HOM}^{-1}) = \mathrm{TC}^0$, where HOM^{-1} denotes inverse morphisms,

[⋆] On sabbatical leave from the Université de Montréal until August 1998.

yielding yet another characterization of this important subclass of NC^1. In the variable case, the problem **eval** remains in TC^0 while the **range** and **inv** problems capture complexity classes between L and NL.

Problem	*Fixed* setting	*Variable* setting
evaluation	isometric: NC^0 nonisom.: TC^0-complete	isometric: AC^0 nonisom.: TC^0-complete
range	any h: NC^1 chosen h: NC^1-complete [1]	$h(\Sigma)$ prefix code: L-complete unrestricted h: NL-complete
inversion	NC^1	$(\text{Functional-L})^{NL}$

Here we do not distinguish between a circuit-based language class and its functional counterpart. A morphism $h : \Sigma^* \rightarrow \Delta^*$ is isometric iff h applied to each $a \in \Sigma$ yields a word of the same length.

An important part of our results, motivated by recent interest in classes intermediate between L and NL [2,12,14], is the investigation of the problem **range** in the variable case. Restricting the underlying morphism affects the complexity of the **range** problem, e.g. the **range** problem for prefix codes is in L and is complete for this class. Now, one might expect that imposing the code property on $h(\Sigma)$ should render the **range** problem complete for an unambiguous logspace class. However the resulting problem remains NL-complete. We therefore develop properties of codes and of sets of words, in particular the *stratification* property (see Section 4.2), which yield **range** subproblems of complexity identical to the complexity of the graph accessibility problems introduced to study unambiguous logspace classes (see Section 2). In particular, we show that the problem **range** in which $h(\Sigma)$ is a *stratified code* is many-one equivalent to the GAP problem L_{stu} capturing $StUSPACE(\log n)$, and that a variant of **range** in which $h(\Sigma)$ is a *stratified left partial* code is $RUSPACE(\log n)$-complete. This is particularly interesting since $RUSPACE(\log n)$ was only recently found to have a complete problem [12].

Due to space restrictions many of our constructions cannot be given in detail. In particular, the proofs of corollary 3 and theorems 4, 7, 8, 9, and 10 have to be postponed to a full version of this paper.

2 Preliminaries

2.1 Complexity Theory

We assume familiarity with basic complexity theory. In particular, recall $NC^0 \subset AC^0 \subset TC^0 \subseteq NC^1 \subseteq L \subseteq StUSPACE(\log n) \subseteq RUSPACE(\log n) \subseteq UL \subseteq NL \subseteq P \subseteq NP$, where UL is the set of languages accepted by logspace Turing machines which are nondeterministic with at most one accepting computation, $RUSPACE(\log n)$ is defined like UL with the stronger condition that, on any input, at most one path should exist from the initial configuration to *any* accessible configuration y, and $StUSPACE(\log n)$ is defined like UL with yet the stronger condition that, between any pair of configurations (x, y), at most one path should exist from x to y [12]. Furthermore, Functional-L, also denoted FL, is the functional counterpart of L, i.e. the set of functions computable by deterministic Turing machines and FL^{NL} is the set of functions computable by a deterministic Turing machine M having access to an NL-oracle.

In the usual way, DLOGTIME uniformity of a circuit family refers to the ability for a Turing machine equipped with an index tape allowing direct access to its input to recognize in time $O(\log n)$ the extended connectivity language of a circuit. For precise details on circuit descriptions, see [4].

Just as GAP is NL-complete and outdegree-one GAP is L-complete [9], the obvious GAP problems L_{stu}, L_{ru}, and L_u, which are respectively $StUSPACE(\log n)$-hard, $RUSPACE(\log n)$-hard, and UL-hard, were introduced for topologically sorted graphs in [2,12]. L_{ru} is $RUSPACE(\log n)$-complete [12] while L_{stu} is not known to belong to $StUSPACE(\log n)$. Using the closure under complement of NL L_u is already NL-complete.

We will make use of the reducibilities $\leq_m^{\mathrm{AC}^0}$ and $\leq_T^{\mathrm{AC}^0}$, which refer to many-one and Turing AC^0 reducibilities respectively.

2.2 Problem Definitions

Fix a morphism $h : \Sigma^* \to \Delta^*$ for finite alphabets Σ and Δ, with Σ assumed ordered. In the fixed setting, the three problems of interest in this note are:

eval(h) Given $v \in \Sigma^*$, compute $h(v)$. The *decision problem* has $b \in \Sigma$ and $j \in N$ as further inputs, and asks whether b is the jth symbol in $h(v)$.

range(h) Given $w \in \Delta^*$, determine whether $w \in h(\Sigma^*)$.

inv(h) Given $w \in \Delta^*$, express w as $h(a_{i_1})h(a_{i_2})\cdots h(a_{i_k})w'$ such that, first, $|w'|$ is minimal, and second, $v := a_{i_1}a_{i_2}\cdots a_{i_k}$ is lexicographically minimal with respect to the ordering of Σ. The *decision problem* is obtained by adding $j \in N$ as an input parameter, and asking for the jth bit in the representation of w.

In the variable setting, the three problems of interest are **eval**, **range**, and **inv**, defined as above, except that the alphabets Σ and Δ, and the morphism h, now form part of the input.

Fix a finite alphabet Γ. Let $v = a_1a_2\cdots a_n, a_i \in \Gamma, n \geq 0$. The length of v is written $|v|$, and $\#_a(v)$ represents the number of occurrences of $a \in \Gamma$ in v. For $0 \leq i \leq j \leq n$, we define ${}_iv_j$ as $a_{i+1}\cdots a_j$. In particular, ${}_{i-1}v_i$ is a_i if $i \geq 1$.

We encode our problems, in both the fixed and the variable settings, as binary strings. In the fixed setting, a word v over an alphabet Γ is encoded as a sequence of equal length "bytes", each representing a letter in Γ. By the predicate $Q_a(v, i)$, $a \in \Gamma$, $v \in \Gamma^*$, we mean that ${}_{i-1}v_i$ is a (with $Q_a(v, i)$ false when $i > |v|$). In the variable setting, we write $Q_a(v, i)$ as $Q(a, v, i)$ in view of the input-dependent alphabet Σ. Any encoding which allows computing $Q(a, v, i)$ in AC^0 suffices.

To encode a morphism $h : \Sigma^* \to \Delta^*$ (only required in the variable setting), it suffices that the predicate associated with h, denoted $H(b, a, j)$, where $b \in \Delta$, $a \in \Sigma$, and $j \geq 0$, meaning that ${}_{j-1}h(a)_j$ is b, be AC^0-computable. For $h : \Sigma^* \to \Delta^*$, we define $\max(h)$ as $\max\{|h(a)| : a \in \Sigma\}$. The predicate $H(b, a, j)$ extends to $H(b, v, j)$ for $v \in \Sigma^*$ in the obvious way, and computing this extension is the object of the **eval** problem. We define $H(b, v, j)$ to be false if $j > |h(v)|$.

3 Fixed Case

Throughout this section we fix $h : \Sigma^* \to \Delta^*$. We write the associated predicate $H(b, v, j)$, $b \in \Delta$, $v \in \Sigma^*$, $j \in N$ as $H_b(v, j)$, in view of the fixed alphabet Σ.

3.1 Evaluation

The complexity of evaluating h is low, but it depends in an interesting way on whether h is *isometric*, i.e. whether the length of any $h(w)$ only depends on $|w|$.

Proposition 1. **eval**$(h) \in$ *Functional-NC*^{0} *if h is isometric.*

Proof. Let h be isometric and let $c = |h(a)|$ $(= \max(h))$ for any $a \in \Sigma$. Then, for $v \in \Sigma^*$, $b \in \Delta$, and $j \in N$, we have $H_b(v, j) \Leftrightarrow (\exists a \in \Sigma)[Q_a(v, \lceil j/c \rceil) \wedge H_b(a, j \textit{ mod } c)]$. Since Σ is fixed, the existential quantification over a can be done in NC^0. Hence, for each j, $1 \leq j \leq c \cdot |v|$, there is an NC^0 subcircuit C_j computing the jth symbol in $h(v)$. The circuit for **eval**(h) is a parallel arrangement of these C_j, $1 \leq j \leq c \cdot |v|$. Uniformity will be argued in a full version of this paper.

It follows that, when h is isometric, the decision problem **eval**$(h) \in$ AC^0. Interestingly, the converse also holds: if h is not isometric, then **eval**$(h) \notin$ Functional-AC^0 and the decision problem **eval**$(h) \notin$ AC^0, as follows from:

Theorem 1. *If h is non-isometric, then the decision problem* **eval***(h) is TC^0-hard under $\leq_T^{AC^0}$.*

Proof. Since h is not isometric, there exist $a, b \in \Sigma$, $s, t \in N$, such that $|h(a)| = s$ and $|h(b)| = s + t$ with $t > 0$. We claim that MAJORITY, i.e. the language of all words v over the alphabet $\{a, b\}$ having $|v|/2 \leq \#_b(v)$, $\leq_T^{\mathrm{AC}^0}$-reduces to computing the length of $h(v)$. This implies that the extended predicate $H_b(v, j)$, $v \in \Sigma^*$, is TC^0-hard, because $|h(v)|$ is trivially expressed as the largest j, $1 \leq j \leq |v| \cdot \max(h)$, such that $\bigvee_{b \in \Sigma} H_b(v, j)$. And computing the extended predicate $H_b(v, j)$ is precisely the decision problem **eval**(h).

To prove the claim that MAJORITY reduces to computing $|h(v)|$, note that $|v|/2 \leq \#_b(v)$ iff $s \cdot |v| + t \cdot |v|/2 \leq s \cdot |v| + t \cdot \#_b(v)$. Now the left hand side of the latter equality can be computed in AC^0 because s and t are constants. As to the right hand side, it is precisely $|h(v)| = s \cdot \#_a(v) + (s+t) \cdot \#_b(v) = s \cdot |v| + t \cdot \#_b(v)$. Hence one can test $|v|/2 \leq \#_b(v)$ in AC^0, once an oracle gate provides the value of $|h(v)|$. Hence MAJORITY $\leq_T^{\mathrm{AC}^0}$-reduces to computing $|h(v)|$.

Corollary 1. *The decision problem* **eval**(h) *is in* AC^0 *iff* h *is isometric.*

On the other hand, it can be shown that even a variable morphism can always be evaluated in TC^0, see subsection 4.1. Hence, the reduction from MAJORITY to **eval**(h) exhibited in Theorem 1 can in fact be reversed:

Theorem 2. *For each morphism* h*, the decision problem* **eval**$(h) \in TC^0$*.*

Corollary 2. *If* h *is a non-isometric morphism, then the decision problem* **eval**(h) *is* TC^0*-complete under* $\leq_T^{AC^0}$*.*

Corollary 3. $TC^0 = Closure(\leq_T^{AC^0}, HOM^{-1})$*.*

3.2 Range Membership

We now turn to the problem of testing whether a word $w \in \Delta^*$ belongs to $h(\Sigma^*)$. Since Σ^* is regular and the regular languages are closed under morphisms, $h(\Sigma^*)$ is regular. Hence **range**$(h) \in \mathrm{NC}^1$. On the other hand, Schützenberger in 1965 constructed the following family of finite biprefix codes over the binary alphabet $\{a, b\}$: $C_n := \{a^n, a^{n-1}ba, a^{n-2}b, ba^{n-1}\} \cup \{a^iba^{n-i-1} \mid 1 \leq i \leq n-3\}$. Schützenberger [16] proved that, for each n, the symmetric group S_n divides (i.e. is an epimorphic image of a submonoid of) the syntactic monoid of the Kleene closure C_n^* of C_n. Using Theorem IX.1.5 of [17] we get the following result. Again, details are left to to a full version of this paper.

Theorem 3. *For every morphism* h*,* **range**$(h) \in NC^1$*. Furthermore, there exists a morphism* h *such that* **range**(h) *is* NC^1*-complete under* $\leq_m^{AC^0}$*.*

3.3 Inversion

Recall the definition of problem **inv**(h). By using **inv**(h) to determine whether $|w'| > 0$, the decision problem **range**(h) reduces to the decision problem **inv**(h), under $\leq_m^{\mathrm{AC}^0}$ if an obvious encoding is chosen. Hence there is an h such that the decision problem **inv**(h) is NC^1-hard (by Theorem 3). Our goal in this section is to show that **inv**(h) is in Functional-NC^1, which holds, clearly, iff the decision problem **inv**(h) is in NC^1.

Suppose that a word $w \in h(\Sigma^*)$. Then if $h(\Sigma)$ were a code, i.e. if it had the property that any word in $h(\Sigma^*)$ is uniquely expressible as a sequence of elements from $h(\Sigma)$, then computing $h^{-1}(w)$ would simply require computing all the positions i, $1 \leq i \leq |w|$, at which w splits into $w = \alpha\beta$ with $\alpha, \beta \in h(\Sigma^*)$. This set of positions would uniquely break w up into words from $h(\Sigma)$. Since these positions can be computed in NC^1 because $h(\Sigma^*)$ is a regular language, this strategy would solve such instances of **inv**(h) in NC^1. Interestingly, we can combine this strategy with a greedy approach and solve the general case in which $h(\Sigma)$ is not a code. For lack of space we have omitted the proof of the next theorem.

Theorem 4. *For every morphism* h*,* **inv**(h) *is in Functional-*NC^1*.*

4 Variable Case

4.1 Evaluation

The fixed case construction for **eval**(h) carries over to the variable case **eval**. We must compute the predicate $H(b, v, j)$, i.e. determine whether the jth symbol in $h(v)$ is b. In the following let $\pi(v)$ be the *Parikh* vector of a word v and M_h the growth matrix of the morphism h, i.e. $(M_h)_{ik}$ is the number of b_k symbols in $h(a_i)$. Then, $H(b, v, j)$ holds iff there exists $i \in N$ and $a \in \Sigma$ such that $\pi({}_0v_{i-1})M_h\mathbf{1} < j \leq \pi({}_0v_i)M_h\mathbf{1}$ and $Q(a, v, i)$ and $h(a)_{j-(\pi({}_0v_{i-1})M_h\mathbf{1})} = b$. Now the computation of M_h out of h and the computations of $\pi({}_0v_{i-1})M_h\mathbf{1}$ and $\pi({}_0v_i)M_h\mathbf{1}$ out of v can be done in TC^0. Indeed, the integers involved in these computations are *small*, i.e. their absolute values are polynomial in the input size, and arithmetic with such integers can even be carried out in AC^0 [4, Lemma 10.5]. But the computation of the *Parikh* vectors needs counting and is TC^0-hard. Hence computing $H(b, v, j)$ is TC^0-hard and **eval** $\in$ Functional-TC^0, which we record as:

Theorem 5. *The decision problem* **eval** *is TC^0–complete w.r.t. $\leq_T^{AC^0}$.*

We note that **eval** restricted to isometric morphisms is in AC^0.

4.2 Range Membership

Our problem **range** is precisely the *Concatenation Knapsack Problem* considered by Jenner [8], who showed that:

Theorem 6. *(Jenner)* **range** *is NL–complete.*

Recall that a nonempty set $C \subset \Delta^*$ is a *code* if C freely generates C^*, and that C is a *prefix code* if C has the prefix property, i.e. if no word in C has a proper prefix in C. Define **prefixcoderange** to be the **range** problem restricted to input morphisms $h : \Sigma^* \rightarrow \Delta^*$ such that $h(\Sigma)$ is a prefix code. The easy proof of the following will be given in a full version of this paper.

Theorem 7. **prefixcoderange** *is L–complete.*

In view of Theorems 6 and 7, it is natural to ask for properties of $h(\Sigma)$ allowing the **range** problem to capture concepts between L and NL like symmetry or unambiguity. For instance, one might reasonably expect *codes* to capture unambiguity, and thus the obvious problem **coderange** to be complete for one of the unambiguous space classes between L and NL. But the mere problem of testing for the code property is NL-hard as shown by Rytter [15]. In the following subsection, we introduce a more elaborate approach which is able to capture unambiguity.

4.2.1 Stratified Sets of Words and Morphisms

Definition 1. a) *Let $C \subseteq \Delta^*$ be arbitrary. Define a relation $\rho_C \subseteq \Delta \times \Delta$ as $a\ \rho_C\ b$ iff some word in C contains ab as a subword.*

b) *C is said to be* stratified *if ρ_C forms a unique maximal acyclic chain $a_1\ \rho_C\ a_2\ \rho_C\ \cdots\ \rho_C\ a_n$; in that case, the word $\sigma(C) := a_1 a_2 \cdots a_n \in \Delta^*$ is called the* stratification *of C. (Example: The set $\{a, b, ab, bc\} \subset \{a, b, c\}^*$ is stratified, while $\{a, b\}$, $\{ab, ba\}$ and $\{ab, ac\}$ are not.)*

Let $C \subset \Delta^*$ be stratified, where we assume from now on that such a stratified set makes use of all the letters in Δ. Clearly, any word $x \in C$ is a subword of $\sigma(C)$. Then, any word $w \in \Delta^*$ is uniquely expressible as $w = \alpha_1\alpha_2 \ldots \alpha_k$, where each α_i is a maximal common subword of w and $\sigma(C)$.

Proposition 2. *Let $C \subset \Delta^*$ be a stratified set.*

a) *For any $w \in \Delta^*$, $w \in C^*$ iff its canonical decomposition $w = \alpha_1\alpha_2 \ldots \alpha_k$ into maximal subwords common with $\sigma(C)$ satisfies $\alpha_i \in C^*$ for each i.*

b) *C is a code iff each subword of its stratification $\sigma(C)$ is expressible in at most one way as a concatenation of words from C.*

Testing whether a finite set $C \subset \Delta^*$ is stratified is L-complete. On the other hand, although stratification may seem like an overwhelming restriction, **stratifiedrange**, i.e. the **range** problem for stratified $h(\Sigma)$, is NL-hard:

Theorem 8. stratifiedrange *is NL-complete.*

Proof. To see that **stratifiedrange** is in NL, we first test deterministically in log space whether C is stratified. Then we use the fact that **range** $\in$ NL. The construction used to show NL-hardness is strongly inspired by a proof in [7] but has to be omitted because of lack of space.

Hence, like **coderange**, but for a different reason, **stratifiedrange** is NL-complete and thus does not appear to capture an unambiguous logspace class. It is **stratifiedcoderange**, namely the problem **range** restricted to the case of a stratified code $h(\Sigma)$, which bears a tight relationship to $StUSPACE(\log n)$. Again, we have to postpone the proof of the following theorem to a full version of this paper.

Theorem 9. stratifiedcoderange *and L_{stu} are many-one logspace equivalent.*

Hence **stratifiedcoderange** is $StUSPACE(\log n)$-hard, and although we are unable to claim a complete problem for $StUSPACE(\log n)$, we have come very close, since even the $StUSPACE(\log n)$-hard language L_{stu}, specifically tailored to capture $StUSPACE(\log n)$, is only known to be in $RUSPACE(\log n)$ [12]. On the other hand, we can claim a complete problem for the unambiguous class $RUSPACE(\log n)$, as follows.

Proposition 2 states that the code property of a stratified set C translates into the unique expressibility of every expressible subword of the stratification of C as an element of C^*. Let us relax this condition and define a stratified set C to be a *left partial code* if no prefix of $\sigma(C)$ can be expressed in two ways as an element of C^*. Furthermore, we let **lpcstratification** be the special case of the **leftpartialcoderange** problem in which we are only asked to determine whether $\sigma(h(\Sigma)) \in h(\Sigma^*)$, i.e. **lpcstratification** is the language of all morphisms $h : \Sigma^* \to \Delta^*$ such that $h(\Sigma)$ is a left partial code and the stratification of $h(\Sigma)$ is in $h(\Sigma^*)$. Then

Theorem 10. lpcstratification *is* $RUSPACE(\log n)$*-complete.*

Proof sketch. The construction used in Theorems 8 and 9 in effect proves that **lpcstratification** and L_{ru} are many-one logspace equivalent. Here L_{ru} is the language of graphs having a path from source to target, but required to possess the unique path property only from the source to any accessible node. This is precisely the property which corresponds to the action of canonically expressing the stratification of a left partial code. We then appeal to the main result of [12], namely that L_{ru} is $RUSPACE(\log n)$-complete.

4.3 Inversion

The class (Functional-L)^{NL} can be defined equivalently as the set of functions computed by single valued NL–transducers. We have:

Theorem 11. *Problem* **inv** *is in (Functional-L)*^{NL}.

Proof. To solve **inv**, we use the algorithm and the automaton constructed in Theorem 4. Here, instead of first producing $\widehat{w}$, we start the deterministic logspace simulation of the automaton. Whenever the simulation would require reading a letter from $\widehat{w}$, we appeal to an NL-oracle to test whether the current input position breaks w up into a prefix and a suffix in $h(\Sigma^*)$ (an NL-oracle can test this by Theorem 6). This concludes the proof.

We remark that the deterministic version of **inv**, namely the restriction of **inv** in which h is a prefix code, belongs to Functional-L.

Since computing the **inv** function allows answering the **range** question, we have $\mathrm{NL} \subseteq \mathrm{L}^{\mathbf{inv}}$, and by the previous theorem $\mathrm{NL} = \mathrm{L}^{\mathbf{inv}}$. This implies that $\mathrm{FL}^{\mathrm{NL}} = \mathrm{FL}^{\mathbf{inv}}$, i.e. that the **inv** function is Turing-complete for $\mathrm{FL}^{\mathrm{NL}}$.

References

1. E. Allender, V. Arvind, and M. Mahajan, *Arithmetic Complexity, Kleene Closure, and Formal Power Series,* DIMACS TechRep. 97-61, September 1997. 248
2. E. Allender and K.-J. Lange, *StUSPACE*$(\log n) \subseteq$ *DSPACE*$(\log^2 n/\log\log n)$, *Proc. of the 7th ISAAC*, Springer LNCS vol. 1178, pp. 193–202, 1996. 248, 249
3. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.*, 38:150–164, 1987. 247

4. D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990. 249, 252
5. D. Barrington and D. Thérien, Finite Monoids and the Fine Structure of NC^1, *J. of the ACM*, 35(4):941-952, 1988. 247
6. S. Eilenberg, *Automata, Languages and Machines,* Academic Press, Vol. B, (1976). 247
7. P. Flajolet and J. Steyaert, *Complexity of classes of languages and operators*, Rap. de Recherche 92 de l'IRIA Laboria, novembre 1974. 247, 253
8. B. Jenner, Knapsack problems for NL, *Information Processing Letters* 54 (1995), pp. 169-174. 252
9. N. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–85, 1975. 249
10. J. Håstad, *Computational Limitations for Small-Depth Circuits,* PhD Thesis, M.I.T., ACM Doctoral Dissertation Awards, MIT Press (1987). 247
11. T. Harju and J. Karhumäki, Morphisms, in *Handbook of Formal Languages*, ed. G. Rozenberg and A. Salomaa, Springer, 1997. 247
12. K.-J. Lange, An unambiguous class possessing a complete set, *Proc. 14th Annual Symp. on Theoret. Aspects of Computer Science,* Springer LNCS vol. 1200, pp. 339–350, 1997. 248, 249, 253, 254
13. P. McKenzie, P. Péladeau and D. Thérien, NC^1: The Automata-Theoretic Viewpoint, *Computational Complexity* 1:330-359, 1991. 247
14. K. Reinhardt and E. Allender, Making nondeterminism unambiguous, *Proc. of the 38th IEEE FOCS* , pp. 244–253, 1997. 248
15. W. Rytter, The space complexity of the unique decipherability problem, *Information Processing Letters* 23 (1986), pp. 1–3. 252
16. M. Schützenberger, On finite monoids having only trivial subgroups, *Information and Control* 8, 190–194, 1965. 251
17. H. Straubing, *Finite automata, formal logic, and circuit complexity*, Birkhäuser, Boston, 1994. 251

Characterization of Efficiently Solvable Problems on Distance-Hereditary Graphs

Sun-Yuan Hsieh[1], Chin-Wen Ho[2], Tsan-Sheng Hsu[3], Ming-Tat Ko[3], and Gen-Huey Chen[1]

[1] Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, ROC
[2] Department of Computer Science and Information Engineering
National Central University, Chung-Li, Taiwan, ROC
[3] Institute of Information Science
Academia Sinica, Taipei, Taiwan, ROC

Abstract. In the literature, there are quite a few sequential and parallel algorithms to solve problems in a distance-hereditary graph G utilizing techniques discovered from the properties of the problems. Based on structural properties of G, we first sketch characteristics of problems which can be systematic solved on G and then define a general problem-solving paradigm. Given a decomposition tree representation of G, we propose a unified approach to construct sequential dynamic-programming algorithms for several fundamental graph-theoretical problems that fit into our paradigm. We also show that our sequential solutions can be efficiently parallelized using the tree contraction technique.

1 Introduction

A graph is *distance-hereditary* [2,11] if the distance stays the same between any of two vertices in every connected induced subgraph containing both (where the *distance* between two vertices is the length of a shortest path connecting them). Distance-hereditary graphs form a subclass of perfect graphs [7,10,11] that are graphs G in which the maximum clique size equals the chromatic number for every induced subgraph of G [9]. Two well-known classes of graphs, trees and cographs, both belong to distance-hereditary graphs. Properties of distance-hereditary graphs are studied by many researchers [2,3,4,5,6,7,8,10,11,12,13] which resulted in sequential or parallel algorithms to solve quite a few interesting graph-theoretical problems on this special class of graphs.

Known polynomial time algorithms on distance-hereditary graphs utilize techniques discovered from the properties of the problems. In this paper, we propose a different approach. Given a graph problem, we say it belongs to the class of *subgraph optimization* problem if the object of this problem aims at finding a subgraph of the input graph to satisfy the given properties which includes an optimization constraint. For example, the problem of finding a maximum independent set is a subgraph optimization problem. By discovering fundamental properties of distance-hereditary graphs, we define a general problem-solving paradigm for subgraph optimization problems. Problems fit into our paradigm include (a) the maximum independent set problem, (b) the maximum clique problem, (c) the Steiner tree problem, (d) the vertex connectivity problem, (e) the edge connectivity problem, (f) the dominating set problem, (g) the independent dominating set problem and (h) the connected dominating set problem. We develop a systematic approach to construct dynamic programming algorithms. For the above problems, our constructed algorithm run in $O(n+m)$ time, where n and m are the number of vertices and edges, respectively. In addition to the sequential solutions, we also show that our general problem-solving paradigm can be efficiently parallelized if a binary tree representation of a distance-hereditary graph is given. The parallel complexities of problems (a)–(h) are $O(\log^2 n)$ time using $O(n + m)$ processors on a CREW PRAM. If a binary tree representation is given, all but the problem (c) can be solved optimally in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.

2 Preliminaries

This paper considers finite, simple, undirected graphs $G = (V, E)$, where V and E are the vertex and edge sets of G, respectively. Let $n = |V|$ and $m = |E|$. For standard graph-theoretic terminologies and notations, see [9]. Let $G[X]$ denote the subgraph of G induced by $X \subseteq V$. The *union* of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = (V_1 \cup V_2, E_1 \cup E_2)$. The class of distance-hereditary graphs can be defined by the following recursive definition.

Definition 1. [4] (1) A graph consisting of a single vertex v is a *primitive* distance-hereditary graph with the twin set $\{v\}$. (2) If G_1 and G_2 are distance-hereditary graphs with the twin sets S_1 and S_2, respectively, then the union of G_1 and G_2 is a distance-hereditary graph G with the twin set $S_1 \cup S_2$. In this case, we say G is formed from G_1 and G_2 by *the false twin operation* $\odot$. (3) If G_1 and G_2 are distance-hereditary graphs with the twin sets S_1 and S_2, respectively, then the graph G obtained from G_1 and G_2 by connecting every vertex of S_1 to all vertices of S_2 is a distance-hereditary graph with the twin set $S_1 \cup S_2$. In this case, we say G is formed from G_1 and G_2 by the *true twin operation* $\otimes$. (4) If G_1 and G_2 are distance-hereditary graphs with the twin sets S_1 and S_2, respectively, the graph G obtained from G_1 and G_2 by connecting every vertex of S_1 to all vertices of S_2 is a distance-hereditary graph with the twin set S_1. In this case, we say G is formed from G_1 and G_2 by the *attachment operation* $\oplus$.

The above definition implies that a distance-hereditary graph can be represented by a binary tree form, called a *decomposition tree*. The complexities of constructing a decomposition tree are shown below.

Lemma 1. *[4,14] A decomposition tree of a distance-hereditary graph can be constructed in sequential $O(n + m)$ time, and in parallel $O(\log^2 n)$ time using $O(n + m)$ processors on a CREW PRAM.*

3 A General Problem-Solving Paradigm

Throughout this paper, we assume each distance-hereditary graph under consideration is represented by its decomposition tree. For a vertex x in a decomposition tree of G, let $T(x)$ be the subtree rooted at x and let G_x be the subgraph of G induced by the leaves of $T(x)$. Also let S_x be the twin set of G_x and let $V_x = V(G_x)$. Let H be a subgraph of G and X be a subset of V.

Definition 2. Define $\mathcal{P}^1_X(G, H)$ to be the problem of finding a subgraph Q in H such that the following two properties hold: (1) the graph $G[V \setminus V(Q)]$ satisfies the given property $\mathcal{P}$, (2) there exists a component C in $G[V \setminus V(Q)]$ satisfying $V(C) \cap X = \emptyset$. For convenience, we use $\mathcal{P}^1_X(G)$ to represent $\mathcal{P}^1_X(G, H)$ if $H = G$, and let $P^1_X(G)$ denote a solution of $\mathcal{P}^1_X(G)$. The superscript 1 in the notations used represent the problem under consideration as *type 1*.

Definition 3. Define $\mathcal{P}^2_X(G, H)$ (respectively, $\mathcal{P}^2_{\overline{X}}(G, H)$) to be the problem of finding a subgraph Q in H such that the following two conditions hold: (1) Q satisfies the given property $\mathcal{P}$, (2) there exists $Q' \subseteq Q$ such that for each connected component C_i, $(V(Q') \cap V(C_i)) \cap X \neq \emptyset$ (respectively, $(V(Q') \cap V(C_i)) \cap X = \emptyset$) if $V(Q') \cap V(C_i) \neq \emptyset$ and $X \neq \emptyset$. For convenience, we use $\mathcal{P}^2_X(G)$ (respectively, $\mathcal{P}^2_{\overline{X}}(G)$) to represent $\mathcal{P}^2_X(G, H)$ ($\mathcal{P}^2_{\overline{X}}(G, H)$) if $H = G$, and let $P^2_X(G)$ (respectively, $P^2_{\overline{X}}(G)$) denote a solution of $\mathcal{P}^2_X(G)$ (respectively, $\mathcal{P}^2_{\overline{X}}(G)$). The superscript 2 in the notations used represent the problem under consideration as *type 2*.

Let $A(G)$ and $B_i(G_i)$, $1 \leq i \leq k$, be graph problems with the input graphs G and G_i, respectively. A *reduction scheme* from B_i's to A, denoted by $A(G) \overset{f(n,k)}{\longleftarrow} (B_1(G_1), B_2(G_2), \ldots, B_k(G_k))$ means that a solution of $A(G)$ can be obtained by merging solutions of $B_i(G_i)$'s in $O(f(n, k))$ time, where $n = |V(G)|$. We also say that $A(G)$ is reduced to $B_1(G_1), B_2(G_2), \ldots, B_k(G_k)$. We say the above scheme a *polynomial* (respectively, *constant*) *time reduction scheme* if $O(f(n, k)) = O(n^l)$, l is a constant (respectively, $O(f(n, k)) = O(1)$). For convenience, the superscript $f(n, k)$ in the notation is omitted if $f(n, k) = O(1)$. We also use $f(n)$ to denote $f(n, k)$ if k is a constant. In the following sections, G is a distance-hereditary graph formed from G_1 and G_2 with the twin sets S_1 and S_2, respectively. Let S be the twin set of G.

3.1 Perfectly Dividable Problem and Dividable Problem

Definition 4. Problem $\mathcal{P}^i_\emptyset(G)$, $i \in \{1, 2\}$, is *perfectly dividable* on G if the following two conditions hold. (1) Given $X \in \{\emptyset, S\}$, there exists $Y \in \{\emptyset, S_1\}$ and $Z \in \{\emptyset, S_2\}$ such that $\mathcal{P}^i_X(G) \overset{f(n)}{\longleftarrow} (\mathcal{P}^i_Y(G_1), \mathcal{P}^i_Z(G_2), \mathcal{P}^i_\emptyset(G, G[S_1 \cup S_2]))$. (2) $\mathcal{P}^i_\emptyset(G, G[S_1 \cup S_2])$ can be solved in $O(n^q)$ time, where q is a constant.

Given a decomposition tree, we have the following result based on the dynamic programming technique.

Theorem 1. *Let $\mathcal{P}^i_{\emptyset}(G)$, $i \in \{1,2\}$, be perfectly dividable. $\mathcal{P}^i_{\emptyset}(G)$ can be solved in $O(n \cdot (f(n) + n^q))$ time. Moreover, if $f(n) = O(1)$ and $\mathcal{P}^i_{\emptyset}(G, G[S_1 \cup S_2])$ can be solved in $O(1)$ time, then $\mathcal{P}^i_{\emptyset}(G)$ can be solved in $O(n)$ time.*

Definition 5. Problem $\mathcal{P}^i_{\emptyset}(G)$, $i \in \{1,2\}$, is said to be *dividable* if the following condition holds. Given $X \in \{\emptyset, S, \overline{S}\}$ and $Y \in \{G, G[V \setminus S]\}$, there exist $k = O(1)$ problems B_j's in $\{\mathcal{P}^i_{\emptyset}(G_1), \mathcal{P}^i_{S_1}(G_1), \mathcal{P}^i_{\overline{S_1}}(G_1), \mathcal{P}^i_{\emptyset}(G[V_1 \setminus S_1]), \mathcal{P}^i_{S_1}(G[V_1 \setminus S_1])\}$, and k problems C_j's in $\{\mathcal{P}^i_{\emptyset}(G_2), \mathcal{P}^i_{S_2}(G_2), \mathcal{P}^i_{\overline{S_2}}(G_2), \mathcal{P}^i_{\emptyset}(G[V_2 \setminus S_2]), \mathcal{P}^i_{S_2}(G[V_2 \setminus S_2])$ such that there is a reduction scheme from B_j's and C_j's to $\mathcal{P}^i_X(Y)$.

Theorem 2. *Let $\mathcal{P}^i_{\emptyset}(G)$, $i \in \{1,2\}$, be dividable. $\mathcal{P}^i_{\emptyset}(G)$ can be solved in $O(f(n) \cdot n)$ time. Moreover, if $f(n) = O(1)$, then $\mathcal{P}^i_{\emptyset}(G)$ can be solved in $O(n)$ time.*

3.2 Parallel Implementation of Paradigm

In this section, we define two systems on a given decomposition tree, called closed system A and closed system B, which can be utilized to parallelize the dividable and perfectly dividable problems using the binary tree contraction technique described in [1]. The *binary tree contraction technique* recursively applies two operations, *prune* and *bypass*, to a given binary tree. $Prune(u)$ is an operation which removes a leaf node u from the current tree, and $bypass(v)$ is an operation (following a prune operation) that removes a node v with exactly one child w and then lets the parent of v become the new parent of w.

We define four operators working on graphs. Given a set of subgraphs $H = \{H_1, H_2, \ldots, H_k\}$ of G, where $k \geq 2$, the operator $\textsc{Min}_v$ (respectively, $\textsc{Min}_e$) is applied on H which returns a subgraph in H with the minimum number of the vertices (respectively, edges). The operators $\textsc{Max}_v$ and $\textsc{Max}_e$ are defined similarly.

Definition 6. Let $\mathcal{P}^i_{\emptyset}(G)$, $i \in \{1,2\}$, be a dividable problem and D_G be a decomposition tree. The *closed system A on D_G* is defined as follows. Initially, each leaf l of D_G (representing a primitive distance-hereditary graph G_l) is associated with five solutions of $\mathcal{P}^i_{\emptyset}(G_l)$, $\mathcal{P}^i_{S_l}(G_l)$, $\mathcal{P}^i_{\overline{S_l}}(G_l)$, $\mathcal{P}^i_{\emptyset}(G[V_l \setminus S_l])$, and $\mathcal{P}^i_{S_l}(G[V_l \setminus S_l])$. Assume Φ is a given operator in $\{ \textsc{Min}_v, \textsc{Min}_e, \textsc{Max}_v, \textsc{Max}_e \}$. For each internal node v with the left and right children u and w, respectively, the solutions of $\mathcal{P}^i_{\emptyset}(G_v)$, $\mathcal{P}^i_{S_v}(G_v)$, $\mathcal{P}^i_{\overline{S_v}}(G_v)$, $\mathcal{P}^i_{\emptyset}(G[V_v \setminus S_v])$, and $\mathcal{P}^i_{S_v}(G[V_v \setminus S_v])$ can be obtained by the following rule:

$$P^i_{\emptyset}(G_v) = \Phi\{C^u_{1,1} \cup C^w_{1,1}, C^u_{1,2} \cup C^w_{1,2}, \ldots, C^u_{1,p} \cup C^w_{1,p}\}, \tag{1}$$

$$P^i_{S_v}(G_v) = \Phi\{C^u_{2,1} \cup C^w_{2,1}, C^u_{2,2} \cup C^w_{2,2}, \ldots, C^u_{2,q} \cup C^w_{2,q}\}, \tag{2}$$

$$P^i_{\emptyset}(G[V_v \setminus S_v]) = \Phi\{C^u_{3,1} \cup C^w_{3,1}, C^u_{3,2} \cup C^w_{3,3}, \ldots, C^u_{3,r} \cup C^w_{3,r}\}, \tag{3}$$

$$P^i_{S_v}(G[V_v \setminus S_v]) = \Phi\{C^u_{4,1} \cup C^w_{4,1}, C^u_{4,2} \cup C^w_{4,2}, \ldots, C^u_{4,s} \cup C^w_{4,s}\}, \tag{4}$$

$$P^i_{\overline{S_v}}(G_v) = \Phi\{C^u_{5,1} \cup C^w_{5,1}, C^u_{5,2} \cup C^w_{5,2}, \ldots, C^u_{5,t} \cup C^w_{5,t}\}, \tag{5}$$

where p, q, r, s and t are all constants and $C^u_{j,k} \in \{P^i_\emptyset(G_u), P^i_{S_u}(G_u), P^i_{\overline{S_u}}(G_u),$ $P^i_\emptyset(G[V_u \setminus S_u]), P^i_{S_u}(G[V_u \setminus S_u])\}$ and $C^w_{j,k} \in \{P^i_\emptyset(G_w), P^i_{S_w}(G_w), P^i_{\overline{S_w}}(G_w),$ $P^i_\emptyset(G[V_w \setminus S_w]), P^i_{S_w}(G[V_w \setminus S_w])\}$. Let γ be the root of D_G. The goal is to find a solution of $\mathcal{P}^i_\emptyset(G_\gamma) = \mathcal{P}^i_\emptyset(G)$.

During the process of tree contraction, we construct the following five functions associated with each node $v \in V(D_G)$. Let X_1, X_2, X_3, X_4 and X_5 be indeterminates that stand for the possibly unknown solutions of $\mathcal{P}^i_\emptyset(G_v), \mathcal{P}^i_{S_y}(G_v),$ $\mathcal{P}^i_\emptyset(G[V_v \setminus S_v])$, $\mathcal{P}^i_{S_v}(G[V_v \setminus S_v])$ and $\mathcal{P}^i_{\overline{S_v}}(G_v)$, respectively. The functions associated with v possess the following form: For $1 \le j \le 5$, $F^v_j(X_1, X_2, X_3, X_4, X_5) =$ $\Phi\{X_{a_{j,1}} \cup \alpha_{a_{j,1}}, X_{a_{j,2}} \cup \alpha_{a_{j,2}}, \ldots, X_{a_{j,b_j}} \cup \alpha_{a_{j,b_j}}\}$, where $1 \le a_{j,k} \le 5$ and $\alpha_{a_{j,k}}$ is a subgraph of G_v. We call the above functions *goal functions* and their common form as *closed A form*. Let v' be the child of $par(v)$ which is an ancestor of v or v itself in the original D_G. We let the functions F^v_1, F^v_2, F^v_3, F^v_4 and F^v_5 to represent $P^i_\emptyset(G_{v'})$,$P^i_{S_{v'}}(G_{v'})$, $P^i_\emptyset(G[V_{v'} \setminus S_{v'}])$, $P^i_{S_{v'}}(G[V_{v'} \setminus S_{v'}])$ and $P^i_{\overline{S_{v'}}}(G_{v'})$, respectively.

During the process of executing the tree contraction, some nodes are removed and the schemes of the remaining nodes are adjusted such that the functions remain closed A form and the following invariant is maintained.

Invariant (I): Let v be an internal node of the current tree such that v holds Φ and $\cup$ operators which generate the solutions of $\mathcal{P}^i_\emptyset(G_v), \mathcal{P}^i_{S_v}(G_v), \mathcal{P}^i_\emptyset(G[V_v \setminus S_v])$, $\mathcal{P}^i_{S_v}(G[V_v \setminus S_v])$ and $\mathcal{P}^i_{\overline{S_v}}(G_v)$ based on Equations (1)–(5). Let u (respectively, w) be the left (respectively, right) child of v in the current tree whose associated functions are $F^u_t(X_1, X_2, X_3, X_4, X_5)$ (respectively, $F^w_t(X_1, X_2, X_3, X_4, X_5)$) for all $1 \le t \le 5$. For each $C^u_{k,l} \in \{P^i_\emptyset(G_u), P^i_{S_u}(G_u), P^i_\emptyset(G[V_u \setminus S_u]), P^i_{S_u}(G[V_u \setminus$ $S_u]), P^i_{\overline{S_u}}(G_u)\}$ (respectively, $C^w_{k,l} \in \{P^i_\emptyset(G_w), P^i_{S_w}(G_w), P^i_\emptyset(G[V_w \setminus S_w]),$ $P^i_{S_w}(G[V_w \setminus S_w]), P^i_{\overline{S_w}}(G_w)\}$) in Equations (1)–(5), we replace it with $F^u_{k,l}(X_1, \ldots,$ $X_5) \in \{F^u_1(X_1, \ldots, X_5), F^u_2(X_1, \ldots, X_5), F^u_3(X_1, \ldots, X_5), F^u_4(X_1, \ldots, X_5),$ $F^u_5(X_1, \ldots, X_5)\}$ (respectively, $F^w_{k,l}(X_1, \ldots, X_5) \in \{F^w_1(X_1, \ldots, X_5),$ $F^w_2(X_1, \ldots, X_5), F^w_3(X_1, \ldots, X_5), F^w_4(X_1, \ldots, X_5), F^w_5(X_1, \ldots, X_5)\}$) that represent $C^{u'}_{k,l}$ (respectively, $C^{w'}_{k,l}$), where u' (respectively, w') is the child of v in the original D_G which is an ancestor of u (respectively, w) or u (respectively, w) itself. For brevity, we use $F^u_{j,k}$ (respectively, $F^w_{j,k}$) to denote $F^u_{j,k}(P^i_\emptyset(G_u), P^i_{S_u}(G_u),$ $P^i_\emptyset(G[V_u \setminus S_u]),$
$P^i_{S_u}(G[V_u \setminus S_u]), P^i_{\overline{S_u}}(G_u))$ (respectively, $F^w_{j,k}(P^i_\emptyset(G_w), P^i_{S_w}(G_w), P^i_\emptyset(G[V_w \setminus S_w]),$ $P^i_{S_w}(G[V_w \setminus S_w]), P^i_{\overline{S_w}}(G_w)))$ when there is no confusion. Then, $P^i_\emptyset(G_v) = \Phi\{F^u_{1,1} \cup$ $F^w_{1,1}, F^u_{1,2} \cup F^w_{1,2}, \ldots, F^u_{1,p} \cup F^w_{1,p}\}$; $P^i_{S_v}(G_v) = \Phi\{F^u_{2,1} \cup F^w_{2,1}, F^u_{2,2} \cup F^w_{2,2}, \ldots, F^u_{2,q} \cup$ $F^w_{2,q}\}$; $P^i_\emptyset(G[V_v \setminus S_v]) = \Phi\{F^u_{3,1} \cup F^w_{3,1}, F^u_{3,2} \cup F^w_{3,2}, \ldots, F^u_{3,r} \cup F^w_{3,r}\}$; $P^i_{S_v}(G[V_v \setminus$ $S_v]) = \Phi\{F^u_{4,1} \cup F^w_{4,1}, F^u_{4,2} \cup F^w_{4,2}, \ldots, F^u_{4,s} \cup F^w_{4,s}\}$; $P^i_{\overline{S_v}}(G_v) = \Phi\{F^u_{5,1} \cup F^w_{5,1}, F^u_{5,2} \cup$ $F^w_{5,2}, \ldots, F^u_{5,t} \cup F^w_{5,t}\}$.

Let $G_\emptyset = (\emptyset, \emptyset)$. Initially, we define the functions for a node $v \in D_G$ to be (i) $F^v_1(X_1, X_2, X_3, X_4, X_5) = \Phi\{X_1 \cup \alpha_1\}$,

(ii) $F_2^v(X_1, X_2, X_3, X_4, X_5) = \Phi\{X_2 \cup \alpha_2\}$,
(iii) $F_3^v(X_1, X_2, X_3, X_4, X_5) = \Phi\{X_3 \cup \alpha_3\}$, (iv) $F_4^v(X_1, X_2, X_3, X_4, X_5) = \Phi\{X_4 \cup \alpha_4\}$ and (v) $F_5^v(X_1, X_2, X_3, X_4, X_5) = \Phi\{X_5 \cup \alpha_5\}$, where $\alpha_j = G_\emptyset$.

Lemma 2. *During the process of executing the binary tree contraction to remove some nodes, the goal functions of the remaining nodes can be adjusted in* $O(1)$ *time using one processor such that the Invariant (I) is maintained.*

Proof. The proof is by induction on the number of contraction phases. The Invariant (I) holds trivially on the original D_G along with the initial functions defined on $V(D_G)$. We use the tree contraction algorithm to reduce the given input tree D_G into a three-node tree T' such that solving closed system A on D_G is equivalent to solving closed system A on T'. We augment the prune and bypass operations by constructing corresponding functions so as to maintain the invariant (I).

In the execution of the tree contraction, assume $prune(u)$ and $bypass(par(u))$ are performed consecutively. Assume u is the left child of v (the case of u being the right child can be similarly computed). Let $par(u) = v$ and $sib(u) = w$. Note that u and v are removed; hence, their contributions to the five target solutions computed at node $par(v)$ have to be incorporated into the functions associated with w. Assume, without loss of generality, that v contains the operators Φ and $\cup$ which perform Equations (1)–(5). The five target solutions associated with node v is given by

- $P_\emptyset^i(G_v) = \Phi\{F_{1,1}^u \cup F_{1,1}^w(X_1, X_2, X_3, X_4, X_5), F_{1,2}^u \cup F_{1,2}^w(X_1, X_2, X_3, X_4, X_5), \ldots, F_{1,p}^u(P_\emptyset^i(G_u), P_{S_u}^i(G_u), P_\emptyset^i(G[V_u \setminus S_u]), P_{S_u}^i(G[V_u \setminus S_u]), P_{\overline{S_u}}^i(G_u)) \cup F_{1,p}^w(X_1, X_2, X_3, X_4, X_5))\} = \Phi\{Q_1 \cup \Phi\{X_{a_1} \cup \alpha_{a_1}, X_{a_2} \cup \alpha_{a_2}, \ldots, X_{a_r} \cup \alpha_{a_r}\}, Q_2 \cup \Phi\{X_{b_1} \cup \alpha_{b_1}, X_{b_2} \cup \alpha_{b_2}, \ldots, X_{b_s} \cup \alpha_{b_s}\}, \ldots, Q_p \cup \Phi\{X_{c_1} \cup \alpha_{c_1}, X_{c_2} \cup \alpha_{c_2}, \ldots, X_{c_t} \cup \alpha_{c_t}\}\} = \Phi\{\Phi\{X_{a_1} \cup (Q_1 \cup \alpha_{a_1}), X_{a_2} \cup (Q_1 \cup \alpha_{a_2}), \ldots, X_{a_r} \cup (Q_1 \cup \alpha_{a_r})\}, \Phi\{X_{b_1} \cup (Q_2 \cup \alpha_{b_1}), X_{b_2} \cup (Q_2 \cup \alpha_{b_2}), \ldots, X_{b_s} \cup (Q_2 \cup \alpha_{b_s})\}, \ldots, \Phi\{X_{c_1} \cup (Q_p \cup \alpha_{c_1}), X_{c_2} \cup (Q_p \cup \alpha_{c_2}), \ldots, X_{c_t} \cup (Q_p \cup \alpha_{c_t})\},\}$ (distributive law) $= \Phi\{\Phi\{X_{a_1} \cup \alpha'_{a_1}, X_{a_2} \cup \alpha'_{a_2}, \ldots, X_{a_r} \cup \alpha'_{a_r}\}, \Phi\{X_{b_1} \cup \alpha'_{b_1}, X_{b_2} \cup \alpha'_{b_2}, \ldots, X_{b_s} \cup \alpha'_{b_s}\}, \ldots, \Phi\{X_{c_1} \cup \alpha'_{c_1}, X_{c_2} \cup \alpha'_{c_2}, \ldots, X_{c_t} \cup \alpha'_{c_t}\},\} = \Phi\{X_{a_1} \cup \alpha'_{a_1}, X_{a_2} \cup \alpha'_{a_2}, \ldots, X_{a_r} \cup \alpha'_{a_r}, X_{b_1} \cup \alpha'_{b_1}, X_{b_2} \cup \alpha'_{b_2}, \ldots, X_{b_s} \cup \alpha'_{b_s}, \ldots, X_{c_1} \cup \alpha'_{c_1}, X_{c_2} \cup \alpha'_{c_2}, \ldots, X_{c_t} \cup \alpha'_{c_t}\}$ (associative law) $= \Phi\{X_{d_1} \cup ((\cup_j\{\alpha'_{a_j} \mid a_j = d_1\}) \cup (\cup_j\{\alpha'_{b_j} \mid b_j = d_1\}) \ldots, (\cup_j\{\alpha'_{c_j} \mid c_j = d_1\})), X_{d_2} \cup ((\cup_j\{\alpha'_{a_j} \mid a_j = d_2\}) \cup (\cup_j\{\alpha'_{b_j} \mid b_j = d_2\}) \ldots, (\cup_j\{\alpha'_{c_j} \mid c_j = d_2\})), \ldots X_{d_k} \cup ((\cup_j\{\alpha'_{a_j} \mid a_j = d_k\}) \cup (\cup_j\{\alpha'_{b_j} \mid b_j = d_k\}) \ldots, (\cup_j\{\alpha'_{c_j} \mid c_j = d_k\}))\}$ (communicative and distributive laws) $= \Phi\{X_{d_1} \cup \beta_{d_1}, X_{d_2} \cup \beta_{d_2}, \ldots, X_{d_k} \cup \beta_{d_k}\}$ (associative law), where X_i, $1 \leq i \leq 5$, are unknown target solutions of node w, assuming that the invariant (I) holds before the $prune(u)$ and $bypass(v)$ operation. The following four terms can be simplified similarly. $P_{S_v}^i(G_v) = \Phi\{F_{2,1}^u \cup F_{2,1}^w(X_1, \ldots, X_5), \ldots, F_{2,q}^u \cup F_{2,q}^w(X_1, \ldots, X_5)\}$. $P_\emptyset^i(G[V_v \setminus S_v]) = \Phi\{F_{3,1}^u \cup F_{3,1}^w(X_1, \ldots, X_5), \ldots, F_{3,r}^u \cup F_{3,r}^w(X_1, \ldots, X_5)\}$. $P_{S_v}^i(G[V_v \setminus S_v]) = \Phi\{F_{4,1}^u \cup F_{4,1}^w(X_1, \ldots, X_5), \ldots, F_{4,s}^u \cup F_{4,s}^w(X_1, \ldots, X_5)\}$. $P_{\overline{S_v}}^i(G_v) = \Phi\{F_{5,1}^u \cup F_{5,1}^w(X_1, \ldots, X_5), \ldots, F_{5,t}^u \cup F_{5,t}^w(X_1, \ldots, X_5)\}$.

Hence, the contribution of five target solutions to the node w is given by (i) $F_1^w(X_1, X_2, X_3, X_4, X_5) = F_1^v(P_\emptyset^i(G_v), P_{S_v}^i(G_v), P_\emptyset^i(G[V_v \setminus S_v]), P_{S_v}^i(G[V_v \setminus S_v]), P_{\overline{S_v}}^i(G_v)) = \Phi\{X_{h_1} \cup \alpha_{h_1}, X_{h_2} \cup \alpha_{h_2}, \ldots, X_{h_k} \cup \alpha_{h_k}\}$ (the process of simplifying is similar to simplify the function corresponding to $P_\emptyset^i(G_v)$ describe in last paragraph). The following functions can be further simplified as closed A form. (ii) $F_2^w(X_1, \ldots, X_5) = F_2^v$. (iii) $F_3^w(X_1, \ldots, X_5) = F_3^v$. (iv) $F_4^w(X_1, \ldots, X_5) = F_4^v$. (v) $F_5^w(X_1, \ldots, X_5) = F_5^v$. Therefore, the invariant (I) is then maintained. By the definition of the closed A form and the composition of the goal function, the complexities can be done. □

According to Lemma 2, the closed system A can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. Hence, we have the following theorem.

Theorem 3. *Let $\mathcal{P}_\emptyset^i(G)$, $i \in \{1, 2\}$, be a dividable problem and let Φ be a given operator in $\{$ MIN_v, MIN_e, MAX_v, MAX_e $\}$. $\mathcal{P}_\emptyset^i(G)$ can be solved in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM if the following condition holds. Given $X \in \{\emptyset, S, \overline{S}\}$ and $Y \in \{G, G[V \setminus S]\}$, there exist two equal-cardinality sets B whose each element is in $\{P_\emptyset^i(G_1), P_{S_1}^i(G_1), P_{\overline{S_1}}^i(G_l), P_\emptyset^i(G[V_1 \setminus S_1]), P_{S_1}^i(G[V_1 \setminus S_1])\}$ and C whose each element is in $\{P_\emptyset^i(G_2), P_{S_2}^i(G_2), P_{\overline{S_2}}^i(G_2), P_\emptyset^i(G[V_2 \setminus S_2]), P_{S_2}^i(G[V_2 \setminus S_2])\}$ such that $P_X^i(Y) = \Phi\{B_j \cup C_k |\ B_j \in B; C_k \in C\}$.*

A problem $\mathcal{P}_\emptyset^i(G)$ is said to be *type i dividable problem with the closed A reduction scheme* if it satisfies the condition of Theorem 3.

Definition 7. Let $\mathcal{P}_\emptyset^i(G)$, $i \in \{1, 2\}$, be a dividable problem and D_G be a decomposition tree. The *closed system B on D_G* is defined as follows. Initially, each leaf l of D_G is associated with $P_\emptyset^i(G_l)$ and $P_{S_l}^i(G_l)$, and each internal node v with the left and right children u and w is associated with $P_\emptyset^i(G_v, G[S_u \cup S_w])$. Assume Φ is a given operator in $\{$ MIN_v, MAX_v, MIN_e, MAX_e $\}$. The solutions of $\mathcal{P}_\emptyset^i(G_v)$, $\mathcal{P}_{S_v}^i(G_v)$ can be obtained by the following rule:

$$P_\emptyset^i(G_v) = \Phi\{C_1^u, C_1^w, P_\emptyset(G_v, G[S_u \cup S_w])\}, \tag{6}$$
$$P_{S_v}^i(G_v) = \Phi\{C_2^u, C_2^w, P_\emptyset(G_v, G[S_u \cup S_w])\}, \tag{7}$$

where $C_j^u \in \{P_\emptyset^i(G_u), P_{S_u}^i(G_u)\}$ and $C_k^w \in \{P_\emptyset^i(G_w), P_{S_w}^i(G_w)\}$. Let γ be the root of D_G. The goal is to find a solution of $\mathcal{P}_\emptyset^i(G_\gamma) = \mathcal{P}_\emptyset^i(G)$.

With the technique similar to implement the closed system A, we have the following result.

Theorem 4. *Let $\mathcal{P}_\emptyset^i(G)$, $i \in \{1, 2\}$, be a perfectly dividable problem and Φ be a given operator in $\{$ MIN_v, MIN_e, MAX_v, $\text{MAX}_e\}$. Problem $\mathcal{P}_\emptyset^i(G)$ can be solved in $O(\log^k n)$ time using $O(n/\log n)$ processors on an EREW PRAM if the following two conditions hold. (1) For all nodes $v \in V(D_G)$ with the left child and the right child u and w, respectively, $\mathcal{P}_\emptyset^i(G_v, G[S_u \cup S_w])$ can be solved in $O(\log^k n)$ time, $k \geq 1$, using the tree contraction technique. (2) Given $X \in \{\emptyset, S\}$, $Y \in \{\emptyset, S_1\}$ and $Z \in \{\emptyset, S_2\}$, $P_X^i(G) = \Phi\{P_Y^i(G_1), P_Z^i(G_2), P_\emptyset^i(G, G[S_1 \cup S_2])\}$,* A problem $\mathcal{P}^i_\emptyset(G)$ is said to be a *type i perfectly dividable problem with the closed B reduction scheme* if it satisfies the condition of Theorem 4.

4 Complexities of Perfectly Dividable and Dividable Problems

We first consider the vertex connectivity problem. A *vertex* (respectively, *edge*) *separator* of a graph is a minimal set of vertices (respectively, edges) whose removal make the given graph be disconnected or result in a trivial graph. A *minimum vertex* (respectively, *edge*) *separator* of G is a vertex (respectively, edge) separator with the minimum cardinality. We define the *vertex* (respectively, *edge*) *connectivity problem* $\mathcal{VC}_\emptyset(G)$ (respectively, $\mathcal{EC}_\emptyset(G)$) to be the problem that finds a minimum vertex (respectively, edge) separator Q of G. For a distance-hereditary graph G with the twin set S, a minimal vertex (respectively, edge) separator Q is called S-type if there exists a component H of $G[V \setminus Q]$ such that $V(H) \cap S = \emptyset$. $\mathcal{VC}_S(G)$ (respectively, $\mathcal{EC}_S(G)$) is the problem that finds a minimum S-type vertex (respectively, edge) separator of G.

In the remainder of this section, assume $G = (V, E)$ is a distance-hereditary graph formed from G_1 and G_2 with the twin sets S_1 and S_2, respectively. Due to the space limitation, the proof of the following lemma is omitted.

Lemma 3. *Given a decomposition tree D_G of G, $\mathcal{VC}_\emptyset(G, G[S_1 \cup S_2])$ can be solved in $O(n)$ sequential time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.*

Given an operator Φ in {Min$_v$, Min$_e$, Max$_v$, Max$_e$}, we let $\mathcal{A}(G) \xleftarrow{\Phi} (\mathcal{B}_1(G_1), \mathcal{B}_2(G_2), \ldots, \mathcal{B}_k(G_k))$ to represent $A(G) = \Phi\{B_i | \ 1 \le i \le k\}$ and $\mathcal{A}(G) \xleftarrow{\cup} (\mathcal{B}_1(G_1), \mathcal{B}_2(G_2), \ldots, \mathcal{B}_k(G_k))$ to represent $A(G) = \cup\{B_i | \ 1 \le i \le k\}$.

Theorem 5. *$\mathcal{VC}_\emptyset(G)$ is a type 1 perfectly dividable problem with the closed B reduction scheme. Given a decomposition tree D_G, it can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.*

Proof. We first show $\mathcal{VC}_\emptyset(G)$ is type 1 perfectly dividable. Let $X \in \{\emptyset, S\}$. If $G = G_1 \otimes G_2$, then $\mathcal{VC}_X(G) \xleftarrow{\textsc{Min}_v} (\mathcal{VC}_{S_1}(G_1), \mathcal{VC}_{S_2}(G_2), \mathcal{VC}_\emptyset(G, G[S_1 \cup S_2]))$. If $G = G_1 \oplus G_2$, then $\mathcal{VC}_\emptyset(G) \xleftarrow{\textsc{Min}_v} (\mathcal{VC}_{S_1}(G_1), \mathcal{VC}_{S_2}(G_2), \mathcal{VC}_\emptyset(G, G[S_1 \cup S_2]))$ and $\mathcal{VC}_S(G) \xleftarrow{\textsc{Min}_v} (\mathcal{VC}_{S_1}(G_1), \mathcal{VC}_\emptyset(G_2), \mathcal{VC}_\emptyset(G, G[S_1 \cup S_2]))$. If $G = G_1 \odot G_2$, then $\mathcal{VC}_X(G) \xleftarrow{\textsc{Min}_v} (\mathcal{VC}_X(G_1), \mathcal{VC}_X(G_2), \mathcal{VC}_\emptyset(G, G[S_1 \cup S_2]))$. By Lemma 3, $\mathcal{VC}_\emptyset(G, G[S_1 \cup S_2]))$ can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. From Definition 4, Theorem 1 and Theorem 4 the result holds. □

The following theorem can be obtained similar to the above discussion.

Theorem 6. *$\mathcal{EC}_\emptyset(G)$ is a type 1 perfectly dividable problem with the closed B reduction scheme. Given a decomposition tree D_G, it can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.*

It can be verified that the maximum clique problem and the Steiner tree problems are both type 2 perfectly dividable problems with the closed B reduction scheme. Given a decomposition tree, the problems can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.

An *independent set* of a graph is a subset of its vertices such that no two vertices in the subset are adjacent. The *independent set problem* $\mathcal{I}_\emptyset(G)$ is the problem that aims at finding a maximum cardinality independent set. Let $G = (V, E)$ be a distance-hereditary graph and S be the twin set of G. An *S-type independent set* of G is a maximum independent set of G which contains at least one vertex of S. An *$\overline{S}$-type independent set* of G is an independent set of G which contains no vertex of S. A *minimum S-type* (respectively, *$\overline{S}$-type*) *independent set* of G, denoted by $I_S(G)$ (respectively, $I_{\overline{S}}(G)$), is an S-type (respectively, $\overline{S}$-type) independent set of G with the minimum cardinality. Due to the space limitation, the proofs of the following lemma are omitted.

Lemma 4. *Suppose G is a distance-hereditary graph formed from $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with the twin sets S_1 and S_2, respectively, by the true twin operation and S is the twin set of G. Then, (1) $I(G) = \text{MAX}_v\{I_{\overline{S}}(G), I_S(G)\}$, (2) $I_S(G) = \text{MAX}_v\{I_{S_1}(G_1) \cup I_\emptyset(G[V_2 \setminus S_2]), I_\emptyset(G[V_1 \setminus S_1]) \cup I_{S_2}(G_2)\}$, (3) $I_{\overline{S}}(G) = I_{\overline{S_1}}(G_1) \cup I_{\overline{S_2}}(G_2)$, (4) $I_\emptyset(G[V \setminus S]) = I_\emptyset(G[V_1 \setminus S_1]) \cup I_\emptyset(G[V_2 \setminus S_2])$.*

Lemma 5. *Suppose G is a distance-hereditary graph formed from G_1 and G_2 with the twin sets S_1 and S_2, respectively, by the attachment operation and S is the twin set of G. Then, (1) $I_\emptyset(G) = \text{MAX}_v\{I_{\overline{S}}(G), I_S(G)\}$, (2) $I_S(G) = I_{S_1}(G_1) \cup I_\emptyset(G[V_2 \setminus S_2])$, (3) $I_{\overline{S}}(G) = \text{MAX}_v\{I_\emptyset(G_1) \cup I_{S_2}(G_2), I_{\overline{S_1}}(G_1) \cup I_{\overline{S_2}}(G_2)\}$, (4) $I_\emptyset(G[V \setminus S]) = I_\emptyset(G[V_1 \setminus S_1]) \cup I_{\overline{S_2}}(G_2)$.*

Since no vertex of G_1 is adjacent to any vertex of G_2 if G is formed from G_1 and G_2 by the false twin operation, the following lemma can be obtained.

Lemma 6. *Suppose that G is a distance-hereditary graph formed from G_1 and G_2 with the twin sets S_1 and S_2, respectively, by the false twin operation, and S is the twin set of G. Then, (1) $I_\emptyset(G) = \text{MAX}_v\{I_S(G), I_{\overline{S}}(G)\}$, (2) $I_S(G) = \text{MAX}_v\{I_{S_1}(G_1) \cup I_\emptyset(G_2),\ I_\emptyset(G_1) \cup I_{S_2}(G_2)\}$, (3) $I_{\overline{S}}(G) = I_{\overline{S_1}}(G_1) \cup I_{\overline{S_2}}(G_2)$, (4) $I_\emptyset(G[V \setminus S]) = I_\emptyset(G[V_1 \setminus S_1]) \cup I_\emptyset(G[V_2 \setminus S_2]$.*

Theorem 7. *$\mathcal{I}_\emptyset(G)$ is a type 2 dividable problem with the closed A reduction scheme. Given a decomposition tree D_G, $\mathcal{I}_\emptyset(G)$ can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors an EREW PRAM.*

Proof. By Lemmas 4–6, Definition 5, $\mathcal{I}_\emptyset(G)$ is a type 2 dividable problem. By Theorems 2 and 3, the complexities follow. □

The domination problem, the connected domination problem, the independent domination problem can be verified as type 2 dividable problems with the closed A reduction scheme. Given a decomposition tree D_G, these problems can be solved in linear time and in parallel $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.

References

1. K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm. *Journal of Algorithms.*, 10, pp. 287-302, 1989. 260
2. H. J. Bandelt and H. M. Mulder. Distance-hereditary graphs. *Journal of Combinatorial Theory Series B*, 41(1):182-208, Augest 1989. 257
3. A. Brandstädt and F. F. Dragan, A linear time algorithm for connected γ-domination and Steiner tree on distance-hereditary graphs, *Networks*, 31:177-182, 1998. 257
4. M. S. Chang, S. Y. Hsieh, and G. H. Chen. Dynamic Programming on Distance-Hereditary Graphs. Proceedings of 7th International Symposium on Algorithms and Computation, ISAAC97, to appear. 257, 258
5. E. Dahlhaus, "Optimal (parallel) algorithms for the all-to-all vertices distance problem for certain graph classes," Lecture notes in computer science 726, pp. 60-69. 257
6. E. Dahlhaus. Efficient parallel recognition algorithms of cographs and distance-hereditary graphs. *Discrete applied mathematics*, 57(1):29-44, February 1995. 257
7. A. D'atri and M. Moscarini. Distance-hereditary graphs, steiner trees, and connected domination. *SIAM Journal on Computing*, 17(3):521-538, June, 1988. 257
8. F. F. Dragan, Dominating cliques in distance-hereditary graphs, *Algorithm Theory-SWAT'94 "4th Scandinavian Workshop on Algorithm Theory, LNCS 824, Springer, Berlin*, pp. 370-381, 1994. 257
9. M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, Academic press, New York, 1980. 257, 258
10. P. L. Hammer and F. Maffray. Complete separable graphs. *Discrete applied mathematics*, 27(1):85-99, May 1990. 257
11. E. Howorka. A characterization of distance-hereditary graphs. *Quarterly Journal of Mathematics (Oxford)*, 28(2):417-420. 1977. 257
12. S.-y. Hsieh, C. W. Ho, T.-s. Hsu, M. T. Ko, and G. H. Chen. Efficient parallel algorithms on distance-hereditary graphs. *Parallel Processing Letters*, to appear. A preliminary version of this paper is in *Proceedings of the International Conference on Parallel Processing*, pp. 20–23, 1997. 257
13. S.-y. Hsieh, C. W. Ho, T.-s. Hsu, M. T. Ko, and G. H. Chen. A new simple tree contraction scheme and its application on distance-hereditary graphs. *Proceedings of Irregular'98*, Springer-Verlag, to appear. 257
14. S.-y. Hsieh, Parallel decomposition of Distance-Hereditary Graphs with its application. Manuscript, 1998. 258

Fast Algorithms for Independent Domination and Efficient Domination in Trapezoid Graphs*

Yaw-Ling Lin

Department of Computer Science and Information Management
Providence University,
200 Chung Chi Road, Sa-Lu, Taichung Shang, Taiwan
yllin@csim.pu.edu.tw

Abstract. The weighted independent domination problem in trapezoid graphs was solved in $O(n^2)$ time [1]; the weighted efficient domination problem in trapezoid graphs was solved in $O(n \log\log n + \overline{m})$ time [8], where $\overline{m}$ denotes the number of edges in the complement of the trapezoid graph. In this paper, we show that the minimum weighted independent dominating set and the minimum weighted efficient dominating set in trapezoid graphs can both be found in $O(n \log n)$ time. Both of the algorithms require only $O(n)$ space. Since $\overline{m}$ can be as large as $\Omega(n^2)$, comparing to previous results, our algorithms clearly give more efficient solutions to the related problems.

1 Introduction

The intersection graph of a collection of trapezoids with corner points lying on two parallel lines is called the *trapezoid graph* [5]. The fastest known algorithm for recognition of trapezoid graph is given by Ma and Spinrad in [11] in $O(n^2)$ time. Throughout the paper, we use n (m) to represent the number of vertices (edges) in the given graph. Note that trapezoid graphs are perfect and properly contain both interval graphs and permutation graphs. Trapezoid graphs are perfect since they are cocomparability graphs. Dagan *et al.* [5] show that the channel routing problem is equivalent to the coloring problems on trapezoid graphs and present an $O(n\chi)$ algorithm to solve it where χ is the chromatic number of the trapezoid graph. Felsner *et al.* [6] design $O(n \log n)$ time algorithms for chromatic number, weighted independent set, clique cover and maximum weighted clique for trapezoid graphs. Also, since Steiner [14] show that the Hamiltonian cycle in the cocomparability graph can found in polynomial time, the Hamiltonian cycle problem of trapezoid graphs is also solvable in polynomial time.

A *dominating set* of a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one vertex in D. Each vertex $v \in V$ can be associated with a (non negative) real weight, denoted by $w(v)$. The *weighted domination problem* is to find a dominating set, D, such that its weight $w(D) = \sum_{v \in D} w(v)$ is minimized. A dominating set D is *independent, connected* or *total* if the subgraph induced by D has no edge, is connected, or has no isolated

* Supported in part by the National Science Council, Taiwan, R.O.C, grant NSC-88-2213-E-126-005.

vertex, respectively. Dominating set problem and its variants (discussed later) have many applications in areas like bus routing, communication network, radio broadcast, and social network, ..., etc. [7] An independent dominating set, D, is *efficient* (a.k.a independent perfect dominating set) if every vertex of V is dominated by *exactly* one vertex of D. Efficient domination problem has several interesting applications in coding theory and the resource allocation of parallel processing system [3,8,10].

The decision version of the weighted domination problem is NP-complete even for cocomparability graphs [4]. For trapezoid graphs, Liang [9] shows that the minimum weighted domination and the total domination problem can be solved in $O(mn)$ time. Srinivasan *et al.* [13] shows that the minimum weighted connected domination problem in trapezoid graphs can be solved in $O(m + n \log n)$ time. Arvind *et al.* [1] showed that the weighted independent domination problem in trapezoid graphs can be solved in $O(n^2)$ time. Liang *et al.* [8] showed that the weighted efficient domination problem in trapezoid graphs can be solved in $O(n \log \log n + \overline{m})$ time, where $\overline{m}$ denotes the number of edges in the complement of the trapezoid graph.

The paper is organized as follows. Section 2 establishes basic notations and some interesting properties of trapezoid graphs. Section 3 gives our $O(n \log n)$-time algorithm of finding the minimum weighted independent dominating set in trapezoid graphs. Section 4 extends the algorithm of Section 3 and gives an $O(n \log n)$-time algorithm of finding the minimum weighted efficient dominating set in trapezoid graphs. Finally, we conclude our results in Section 5.

2 Basic Notations and Properties

The *trapezoid representation* of a trapezoid graph $G = (V, E)$ consists of two parallel lines L_1 (the lower line) and L_2 (the upper line), such that for each element $v \in V$ there corresponds a trapezoid using $I_v \subset L_1$ ($J_v \subset L_2$) as the the lower (upper) interval. Counterclockwise rotating the upper line by 90° as the y-axis, and using the lower line as the x-axis, we obtain the *box representation* of a trapezoid graph. Note that each vertex of the trapezoid graph corresponds a rectangle box in the two dimensional representation. Figure 1 shows a trapezoid graph with its corresponding trapezoid and box representation. Clearly the trapezoid representation and the box representation are just two different interpretations of the same problem. Given a vertex $v \in V$, we use $tl(v)$ ($tr(v)$) and $bl(v)$ ($br(v)$) to denote the top left (right) and the bottom left (right) corner points of its corresponding trapezoid representation. Define two points $l(v) = (bl(v), tl(v))$ and $u(v) = (br(v), tr(v))$, both in $\mathbb{R}^2$, be the lower left and upper right corners of the corresponding box of v. It can be shown that any trapezoid representation can be transformed into another trapezoid representation with all corner points are distinct while the two representation still corresponds to the same trapezoid graph. Thus, we assume that each trapezoid of the given problem (with n vertices) has four distinct corners and all corner

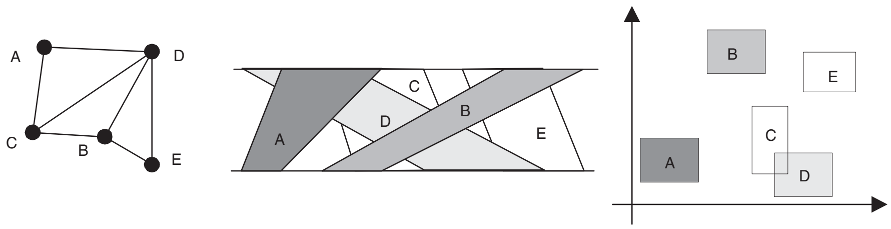


Fig. 1. A trapezoid graph, its trapezoid representation, and the box representation.

points are distinct with consecutive integer values from 1 to $2n$ on both parallel lines.

Given a point $p \in \mathbb{R}^2$, we use p_x (p_y) to denote the x (y) coordinate of p. Given two points $p, q \in \mathbb{R}^2$, p is said to *dominated by* q, denoted by $p < q$, if $p_x < q_x$ and $p_y < q_y$. Similarly, given the box representation of a trapezoid graph, we define two vertices $a < b$ if $u(a) < l(b)$; that is, box(a) lies totally on the left lower side of box(b) or, equivalently, trapezoid(a) lies totally on the left of trapezoid(b). The corresponding partial ordered set is a *trapezoid order*. It can be easily verified that every trapezoid graph G corresponds with a trapezoid order P such that two elements are adjacent in G iff they are incomparable in P. A subset of vertices $U \subset V$ of a given poset $(V, <)$ is called a *chain* if any two vertices of U are comparable. A chain U is *maximal* if no other vertex of V can be added into U to form a larger chain. It is easily verified that

Proposition 1. *Given a trapezoid graph, $G = (V, E)$, a subset of vertices $D \subset V$ is an independent dominating set iff D is a maximal chain in the corresponding trapezoid order.*

Proof. Any two vertices of D are comparable in the trapezoid order since they are independent; further, it is maximal since the other vertices are adjacent to some vertices of D. The converse can also be easily shown. □

Thus, the problem of finding a minimum weighted independent dominating set in a trapezoid graph is equivalent to the problem of finding the minimum weighted maximal chain in the trapezoid order.

Throughout the paper, we use the words "box" and "vertex" interchangingly for convenience. Given a vertex $v \in V$, and a subset of vertices $S \subset V$, define $Dom_S(v) = \{u \in S : u < v\}$; that is, $Dom_S(v)$ are the boxes (vertices) dominated by box(v). A box of S is *maximal* if no other box in S dominates it. We use $Max(S)$ to denote the set of maximal boxes in S; further, we abbreviate $Max(Dom_S(v))$ as $MD_S(v)$; also, $DOM_V(v)$ and $MD_V(v)$ are abbreviated as $DOM(v)$ and $MD(v)$.

The basic idea of our algorithm is the following: we associate with each vertex v an *aggregated weight*, denoted by $w'(v)$. The aggregated weight of v is determined by

$$w'(v) = w(v) + \min\{w'(u) : u \in MD(v)\}$$

The vertex u which gives the minimum $w'(v)$ to v is called the (immediate) *predecessor* of v. To avoid the complicated boundary condition, we insert two dummy vertices (boxes), v_0 located at the $(0, 0)$ and v_{n+1} located at the $(2n + 1, 2n + 1)$ (these two points are the degenerated case of two small boxes), each with zero weight. $w'(v_0)$ is initialized as zero. After computing all the aggregated weights, we have

Proposition 2. *The value of $w'(v_{n+1})$ is the minimum weight of the minimum weighted independent dominating set. Further, the predecessors of v_{n+1} are exactly the minimum weighted independent dominating set of the given trapezoid graph.*

Proof. It is easily seen that the predecessors of v_{n+1} is a maximal chain of the trapezoid order. Further, the last box of the minimum weighted maximal chain is exactly the (immediate) predecessor of v_{n+1}, caught while computing $w'(v)$. The rest of the proof follows easily by induction. □

3 Independent Domination of Trapezoid Graphs

Immediately following the previous discussion, a simple-minded algorithm can be constructed as following: for each vertex v, find the sets $Dom(v)$ and $MD(v)$, and compute $w'(v)$ from left to right. If we do it carefully, it will cost $O(n + \overline{m})$ time to finish. Unfortunately, $\overline{m}$ can be as large as $\Omega(n^2)$. The essential idea for a faster algorithm is that we can not afford to examine every vertex in $DOM(v)$ (or even $MD(v)$) to determine the value of $w'(v)$ for each vertex v.

The general idea of our algorithm is basically a recursively divide-and-conquer scheme. According to the x coordinates of the lower corners of V, we can partition the n boxes into two equal-sized subsets $L, R \subset V$ ($|L| = |R| = n/2$) such that all lower corners of L lie on the left of any lower corner of R; that is, $\max\{l(v)_x : v \in L\} < \min\{l(v)_x : v \in R\}$. First, we recursively solve the problem for boxes in L. A *propagation step* will collect the information computed from the previous step and propagate these information into vertices of R. Finally, another recursive call for boxes in R (which now aggregated with extra information from L) will reveal the final solution. Note that the propagation step takes place *before* the second recursive call. The goal here is to do the propagation step in $O(n)$, implying that the problem can be solved in $O(n \log n)$ time. The Overall scheme described so far is very similiar to [2] for finding the independet dominating set of permuation graphs. However, the propagation step we used in the trapezoid graphs models involves two-dimensional boxes, which will complicate the analysis and the algorithm. In contrast to the permutation graph, the models in consideration are just single points in the two-dimensional plane.

Let S be a subset of boxes in V and let v be a box whose lower corner lies on the right of all lower corners of S. Note that some boxes of $MD(v)$ belong to S. Define $mp_S(v)$ (the minimum weight of the predecessor portion of S) as the minimum weight of boxes in $S \cap MD(v)$; i.e.,

$$mp_S(v) = \begin{cases} \min\{w'(a) : a \in S \cap MD(v)\} & \text{if } S \cap MD(v) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

Further, denote the two boxes with the high-most and low-most *upper* corners in $S \cap MD(v)$ by $first_S(v)$ and $last_S(v)$. Note that $first_S(v)$ or $last_S(v)$ might not exist; in which case, we use $\emptyset$ as the function value.

Note that in the recursive version of our algorithm, the given input will be a set of boxes S, sorted by the consecutive x coordinates of their *lower corners*. Let A be boxes of V that lie on the left hand side of S. While computing $mp_S(v)$, the $w'(\cdot)$ values of boxes in A shall have already been determined by the algorithm. The recursive algorithm will divide S into two equal-sized subsets L and R. After return from the first recursive processing for L, the $w'(\cdot)$ values of each box of L (together with the minimum predecessor links) are now determined.

The goal of the propagation step is, for each $v \in R$, to determine the set values of $first_{A\cup L}(v)$, $last_{A\cup L}(v)$, and $mp_{A\cup L}(v)$ in $O(|S|)$ time during the propagation step. Before we go into the details, observe that our dynamic programming formula for determining the $w'(v)$, $v \in L$, can now be viewed as the following:

$$w'(v) = w(v) + \min\left(\{mp_A(v)\} \cup \{w'(u) : u \in MD_L(v)\}\right)$$

For the clearness of the presentation, we will mainly discuss how the *value* of the minimum weight problem is calculated; however, it shall be clear that we can maintain a *predecessor link* for each vertex $v \in V$ when the final $w'(v)$ is determined. So the corresponding problem for finding the minimum independent dominating *set* can easily be modified in the algorithm. Further, if we choose the terminated condition for the recursive algorithm as when $|S| = 1$, $MD_L(v)$ is just an empty set. In which case, $w'(v)$ is simply given by $w(v) + mp_A(v)$. Thus, if the value of $mp_{A\cup L}(v)$ for each box $v \in R$ is determined in the propagation step, the same recursive scheme can now be applied to the boxes of R (now substitute S by R and substitute A by $A \cup L$.) Further, for each box v in R, the value of $mp_{A\cup L}(v)$ is determined by the following formula:

$$mp_{A\cup L}(v) = \min\{mp_A(v), mp_L(v)\}$$

The situation is illustrated in Figure 2. Also, note that $first_{A\cup L}(v)$ can be determined by the values of $first_A(v)$ and $first_L(v)$. Likewise, $last_{A\cup L}(v)$ can be determined by the values of $last_L(v)$ and $last_A(v)$. It shall be clear now that the essential part of the propagation step is:

Lemma 1. *For each $v \in R$, values of $mp_L(v)$, as well as $first_L(v)$ and $last_L(v)$ can be determined totally within $O(|S|)$ time.*

Proof. From bottom to up, we scan each box of R according to the y coordinates of the lower corners. During the scanning process, we maintain a stack, K,

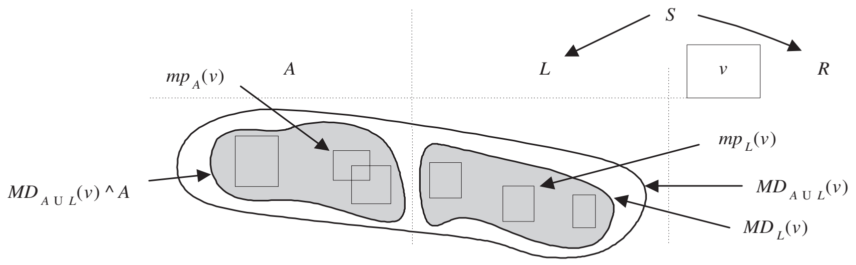


Fig. 2. Calculation of $mp_{A\cup L}(v)$.

containing a subset of R scanned so far. Boxes of K form a (so far maximal) chain in the trapezoid (box) order. That is, if $K = \langle b_i, \ldots, b_1\rangle$ (from bottom to top), then we have $b_i < b_{i-1} < \cdots < b_1$. Identify boxes of L whose y coordinates of the *upper corners* located between $l(b_1)_y$ and $l(v)_y$ but not include those boxes that are not dominated by v. Denote these boxes by B_v. It is not hard to verify that we can find $M_0 = Max_{B_v}(v)$ in $O(|B_v|)$ time [12]. Let $high(M_0)$ and $low(M_0)$ be the two boxes with the high-most and the low-most upper corners of M_0. If the top of the stack, b_1, is dominated by the currently scanned box v, it is clear that M_0 is exactly the set $MD_L(v)$. Thus we can scan boxes of M_0 to compute $mp_L(v) = \min\{w'(a) : a \in M_0\}$. Note that $first_L(v)$ and $last_L(v)$ are just $high(M_0)$ and $low(M_0)$. The box v is pushed into the stack when $mp_L(v)$, $first_L(v)$, and $last_L(v)$ are determined.

For the case that b_1 is *not* dominated by v, we can pop out b_1 and check for the condition of b_2. Finally, we shall find a box of K, b_j, that is dominated by v. (Recall that we can add a dummy box in the bottom of stack to avoid the complicated boundary condition.) Note that $b_j < v$, but boxes $b_1, \ldots, b_{j-1}$ are not dominated by v. Similar to the definition of M_0, there shall be a sequence of subsets of L, namely $M_1 = MD_L(b_1), \ldots, M_{j-1} = MD_L(b_{j-1})$; they are determined while boxes $b_1, \ldots, b_{j-1}$ being pushed into the stack. The set $MD_L(v)$ can now be determined by scanning through all vertices of $M_0 \cup \cdots \cup M_{j-1}$; it follows that $mp_L(v) = \min\{w'(a) : a \in MD_L(v)\}$, and the corresponding $first_L(v)$ and $last_L(v)$ can also be determined. Again, we will then push v into the stack. It shall be clear that, after the topmost box of R is scanned, we have correctly determined the $mp_L(\cdot)$, $first_L(\cdot)$, and $last_L(\cdot)$ values for each box of R.

To justify the whole process takes $O(|S|)$ time, first observe that the computation of $M_0 = Max_{B_v}(v)$ for each box $v \in R$ totally spends $O(|R|) + \sum_{v\in R} O(|B_v|) = O(|R|) + O(|L|)$ time. Secondly, every box, in R, that is pop out of the stack will not be reexamined again, so the total cost is $O(|R|)$. Most importantly, since boxes of M_i's are sorted, from high to low, according to the y coordinates of their upper corners, the process of merging all M_i's into $MD_L(v)$ takes $O(j)$ time plus the time needed for eliminating the unwanted boxes (they are dominated by some boxes of M_k who lies above them.) The important observation is that these boxes can only be eliminated once because they are dominated by boxes of some M_k. Since $\sum O(j) = O(|R|)$, the time needed to maintain the stack, the cost of elimination totally takes only $O(|L|)+O(|R|)$ time. Thus we conclude that the propagation step of the algorithm spends $O(|L|+|R|) = O(|S|)$ time. □

Now we are ready to present our algorithm for finding the minimum independent dominating set of trapezoid graphs as shown in Figure 3.

```
ALGORITHM MIDS($S$)
Input: A sequence of contiguous boxes $S = \langle v_i, \ldots, v_{i+\ell-1} \rangle$ of $V$. Note that
the values of $mp_A(\cdot)$ have already been determined where $A = \langle v_0, \ldots, v_{i-1} \rangle$.
Output: The values of $w'(\cdot)$ for each box in $S$.
Step 1: If $\ell = 1$, $w'(v_i) = w(v_i) + mp_A(v_i)$; return to the caller.
Step 2: Split $S$ into two equal-sized subsequence $L$ and $R$.
Step 3: Recursively call MIDS($L$), which computes values of $w'(\cdot)$ for boxes
in $L$.
Step 4: The propagation step, described in Lemma 1 will modify the values of
$mp_{A \cup L}(\cdot)$ for each vertex in $R$ accordingly.
Step 5: Recursively call MIDS($R$), which computes values of $w'(\cdot)$ for boxes
in $R$.
END OF MIDS
```

Fig. 3. finding the minimum independent dominating set in trapezoid graphs.

Theorem 1. *Let v_0 and v_{n+1} be two zero-weighted dummy boxes, located at the $(0,0)$ and $(2n+1, 2n+1)$ (these two boxes are two degenerated boxes as two points.) The algorithm MIDS($\{v_0 \cup V \cup \{v_{n+1}\}$) return $w'(\cdot)$ for each vertex in V in $O(n \log n)$ time and $O(n)$ space. Further, $w'(v_{n+1})$ is the solution for the minimum weighted independent dominating set problem.*

Proof. The correctness of the algorithm follows from Observation 2 and Lemma 1. Let $T(\ell)$ denote the running time of the algorithm MIDS(S). Clearly Step 2 requires at most $O(\ell)$ time. Further, by Lemma 1, Step 4 requires $O(\ell)$ time. Thus we have the recurrence $T(\ell) = 2T(\ell/2) + O(\ell)$. It follows that $T(\ell) = O(\ell \log \ell)$. Further, space requirement for the algorithm is for the values of $w'(\cdot), mp(\cdot), first(\cdot)$, and $last(\cdot)$'s for each box of V. Thus we conclude that MIDS($\{v_0 \cup V \cup \{v_{n+1}\}$) correctly determines the minimum weighted independent dominating set problem in $O(n \log n)$ time and $O(n)$ space. □

4 Efficient Domination of Trapezoid Graphs

As mentioned in Section 1, efficient domination problem has several interesting applications in coding theory and the resource allocation of parallel processing system. Here we present our algorithm, which is very similiar to the algorithm mentioned at the previous section, for finding the weighted efficient dominating set of trapezoid graphs in $O(n \log n)$ time.

By Observation 1, we know the efficient dominating set of a trapezoid graph forms a maximal chain in the trapezoid order since an efficient dominating set is an independent dominating set. Thus, it is possible that we can refine the MIDS algorithm to solve the efficient domination problem in trapezoid graphs.

Given a box v, consider all boxes, A, whose *top left* corners lie on the top left position of the *lower corner* of v. Let $t(v)$ be the box whose top left corner is the leftmost point among A. Symmetrically, consider all boxes, B, whose *bottom right* corners lie on the bottom right position of the *upper corner* of v. Let $b(v)$ be the box whose top left corner is the rightmost point among B. It is shown that $t(v)$ and $b(v)$ can be computed in $O(n)$ time [8]. Further, we denote the top left corner (a point) of $t(v)$ by $t'(v)$, and the bottom right corner of $b(v)$ by $b'(v)$. Note that, if u is an immediate predecessor of v ($u \in MD(v)$), u might not be an immediate *efficient predecessor* of v. That is, there might be a box incomparable to both u and v. However, it is easily verify that u is also an efficient predecessor of v if v dominates $b(u)$ and, at the same time, $t(v)$ dominates u. Note that both of the conditions can be checked in constant time after the $t(\cdot), b(\cdot)$ values are decided.

Note that the only place we need to change in the algorithm MIDS is to modify the meaning of $MD(v)$ (immediate predecessors) into $MD'(v)$ (immediate efficient predecessors.) That is, in the proof of Lemma 3, when the algorithm try to find the set $M_0 = Max_{B_v}(v)$, we also check the condition that whether boxes of $Max_{B_v}(v)$ are efficient predecessors of v. Only the efficient predecessors in Max_{B_v} are put into M_0.

The correctness and time complexity of the modification version of the algorithm, which we call it the MEDS algorithm for the obvious reason, is similar to what we have done for the MIDS algorithm, and is skipped here.

Theorem 2. *The modified algorithm MEDS(*$\{v_0 \cup V \cup \{v_{n+1}\}$*) return* $w'(\cdot)$ *for each vertex in* V *in* $O(n \log n)$ *time and* $O(n)$ *space. Further,* $w'(v_{n+1})$ *is the solution for the minimum weighted efficient dominating set problem.*

5 Concluding Remarks

In this paper, we show that the minimum weighted independent dominating set problem and the minimum weighted efficient dominating set problem in trapezoid graphs can both be efficiently solved in $O(n \log n)$ time. It is interesting to note that, since the essential scheme of our algorithm a recursive one, the algorithm does not benefit from the fact that the input trapezoid representation are given in the consecutive integer values. Is it possible that the minimum weighted independent dominating set problem of trapezoid graphs (and permutation graphs as well) can be solved in time complexity like $O(n \log \log n)$? Finally, the author will like to thank Chin-Lung Lu for many valuable discussions of the efficient domination problem.

References

1. K. Arvind and C. Pandu Rangan. Efficient algorithms for domination problems on cocomparability graphs. Manuscript, 1990. 267, 268
2. M. J. Atallah and S. R. Kosaraju. An efficient algorithm for maxdominance, with applications. *Algorithmica*, 4:221–236, 1989. 270
3. N. Biggs. Perfect codes in graphs. *J. Combin. Theory Ser. B*, 15:289–296, 1973. 268
4. M.-S. Chang. Weighted domination of cocomparability graphs. *Discr. Applied Math.*, 80:135–148, 1997. 268
5. I. Dagan, M.C. Golumbic, and R.Y. Pinter. Trapezoid graphs and their coloring. *Discr. Applied Math.*, 21:35–46, 1988. 267
6. Felsner, Muller, and Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discr. Applied Math.*, 74, 1997. 267
7. T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, Inc., N. Y., 1998. 268
8. Y. D. Liang, C.-L. Lu, and C.-Y. Tang. Efficient domination on permutation graphs and trapezoid graphs. In *COCOON'97*, pages 232–241, 1997. 267, 268, 274
9. Y. Daniel Liang. Dominations in trapezoid graphs. *Information Processing Letters*, 52(6):309–315, December 1994. 268
10. M. Livingston and Q.F. Stout. Perfect dominating sets. *Congr. Numer.*, 79:187–203, 1990. 268
11. T.-H. Ma and J.P. Spinrad. An $O(n^2)$ time algorithm for the 2-chain cover problem and related problems. In *Proc. 2nd ACM-SIAM Symp. Discrete Algorithms*, pages 363–372, 1991. 267
12. M. H. Overmars and J. van Leeuwen. Maintenance of configuration in the plane. *Journal of Computer and Systems Science*, 23:166–204, 1981. 272
13. Anand Srinivasan, M.S. Chang, K. Madhukar, and C. Pandu Rangan. Efficient algorithms for the weighted domination problems on trapezoid graphs. Manuscript, 1996. 268
14. G. Steiner. Polynomial time algorithm for finding a Hamiltonian cycle in the incomparability graph. Manuscript, 1992. 267

Finding Planar Geometric Automorphisms in Planar Graphs[⋆]

(Extended Abstract)

Seok-Hee Hong[1], Peter Eades[2], and Sang-Ho Lee[1]

[1] Department of Computer Science and Engineering
Ewha Womans University, Korea
{shhong,shlee}@cs.ewha.ac.kr

[2] Department of Computer Science and Software Engineering
University of Newcastle, Australia
eades@cs.newcastle.edu.au

Abstract. Drawing a graph symmetrically enables an understanding of the entire graph to be built up from that of smaller subgraphs. This paper discusses symmetric drawings of planar graphs. More specifically, we discuss *planar geometric automorphisms*, that is, automorphisms of a graph G that can be represented as symmetries of a planar drawing of G. Finding planar geometric automorphisms is the first and most difficult step in constructing planar symmetric drawings of graphs. The problem of determining whether a given graph has a nontrivial geometric automorphism is NP-complete for general graphs. In this paper, we present a polynomial time algorithm for finding planar geometric automorphisms of graphs.

1 Introduction

Graph drawing is constructing a visually-informative drawing of an abstract graph in the plane. Symmetry is one of the most important aesthetic criteria that represent the structure and properties of a graph visually. Also, a symmetric drawing of a graph enables an understanding of the entire graph to be built up from that of a smaller subgraph [13,14,15].

Symmetric drawings of a graph G are clearly related to the automorphisms of G (the relationship is discussed below), and algorithms for constructing symmetric drawings have two steps:

1. Find the "appropriate" automorphisms, and
2. draw the graph displaying these automorphisms as symmetries.

⋆ This research has been supported by KOSEF No.971-0907-045-1, the Australian Research Council, and the SCARE project at the University of Limerick. This paper was partially written when the first author was visiting the University of Newcastle, and partially written while the first two authors were visiting the University of Limerick.

Section 2.2 of this paper defines the "geometric automorphisms" of a graph, which are the "appropriate" automorphisms mentioned in the first step above. Intuitively, an automorphism of a graph G is "geometric" if it can be represented as a symmetry of a drawing of G. The concept of geometric automorphism can be refined to the concept of "planar geometric automorphism". We outline an algorithm for finding planar geometric automorphisms. The second step, constructing a drawing which displays the planar geometric automorphisms, is not discussed in this paper.

We should note that the problem of finding a planar geometric automorphism is related to, but not equivalent to, the problem of finding automorphisms. The problem of determining whether a graph has a nontrivial automorphism is equivalent to the graph isomorphism problem, that is, it is *isomorphism complete* [16]. However, the problem of determining whether a graph has a nontrivial *geometric* automorphism is *NP-complete* [12,15]; it is probably strictly harder than graph isomorphism. Previous work on symmetric graph drawing [13,14,15,6] has focused on restricted classes of graphs such as trees [13], outerplanar graphs [14], and series-parallel graphs [6]. This paper presents a polynomial time algorithm for finding planar geometric automorphisms of connected planar graphs.

In the next section, we review the background: methods for finding automorphisms of planar graphs, symmetries of drawings of graphs, and the concept of planar geometric automorphisms. Section 3 outlines our algorithm. Section 4 concludes.

2 Background

2.1 Automorphisms

An *isomorphism* between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a one-one function α from V_1 onto V_2 which preserves adjacency; that is, $(u, v) \in E_1$ if and only if $(\alpha(u), \alpha(v)) \in E_2$. If $G_1 = G_2$ then α is an *automorphism*. The set of automorphisms of a graph forms a group, and a subgroup of this group is an *automorphism group*. The *isomorphism problem* is to determine all isomorphisms between two given graphs G_1 and G_2. If $G_1 = G_2$ then the isomorphism problem is the *automorphism problem*.

Automorphisms of a graph are permutations of the vertices, and some of the terminology of permutation groups is helpful [19]. We denote the identity permutation by I. The group generated by $\alpha_1, \alpha_2, \ldots, \alpha_k$ is denoted by $< \alpha_1, \alpha_2, \ldots, \alpha_k >$. If a permutation α acting on a set V has a *fixed element* $v \in V$, that is, $\alpha(v) = v$, then α induces a permutation α_v on $V - \{v\}$.

The algorithm of Hopcroft and Tarjan [7,8] may be used to compute automorphisms of planar graphs. It proceeds by reducing the problem to the triconnected case; we adopt a similar approach. The algorithm uses $O(n \log n)$ time. Later, Hopcroft and Wong [10] reduced the time complexity to linear, but with a large constant. Fontet [4] presents another linear time algorithm. Most of this research focuses on the case of planar embeddings.

These methods all produce "automorphism partitions" as part of the isomorphism test; essentially the vertex set is partitioned into the orbits of the automorphism group. This can be used to assign an integer $code(G)$ for a planar graph G such that $code(G) = code(G')$ if and only if G is isomorphic to G'. The integer $code(G)$ is called the *isomorphism code* of G. The automorphism partition is useful for drawing graphs symmetrically; intuitively, if G_1 and G_2 are subgraphs of G such that G_1 is isomorphic to G_2, then we can draw G_1 congruently to G_2.

Theorem 1. *[10,4] Suppose that $G_1, G_2, \ldots, G_k$ are planar graphs. Then in linear time we can compute integers $code(G_1), code(G_2), \ldots, code(G_k)$ such that $code(G_i) = code(G_j)$ if and only if G_i is isomorphic to G_j.*

The automorphism partition works even if the graphs are labeled (that is, a vertex of label x can be mapped only to a vertex of label x).

Connectivity plays a large role in these algorithms. The uniqueness of sphere embeddings of triconnected graphs restricts the automorphism group considerably, and makes the problem easier. In general, a planar graph can have an exponential number of planar embeddings. The most difficult parts of algorithms to find automorphisms are the transformations from the triconnected case to the biconnected case, and from the biconnected case to the connected case.

2.2 Symmetries and Geometric Automorphisms

It is important to use a rigorous model for the intuitive concept of symmetry display. The model below is derived from those introduced by Manning [13,14,15] and Lin [11,3].

The symmetries of a set of points in the plane (such as a two dimensional graph drawing) form a group called the *symmetry group* of the set. A symmetry σ of a drawing D of a graph G *induces* an automorphism of G; the restriction of σ to the points representing vertices of G is an automorphism of G. A drawing D of a graph G *displays* an automorphism α of G if there is symmetry σ of D which induces α. The symmetry group of a graph drawing induces an automorphism group of the graph. An automorphism group P of a graph G is a *geometric automorphism group* if there is a drawing of G which displays every element of P.

To understand the difference between automorphisms and geometric automorphisms, consider Figure 1 (from [11,3]). Figure 1(a) shows the graph $K_{2,3}$. The permutation (123)(45) is an automorphism but not a geometric automorphism; in fact the automorphism group of $K_{2,3}$ has size 12, but the only geometric automorphism is the one displayed in Figure 1(a), that is, (13)(2)(45).

A group-theoretic characterization of geometric automorphism groups [11,3] follows. A permutation group P is *semiregular* if each non-identity permutation in P does not have a fixed element. A permutation p on V is a *rotational* permutation if either $< p >$ or $< p_v >$ (for some $v \in V$) is semiregular, and $| < p > | > 1$. Note that a rotational permutation has at most one fixed element.

A permutation p on V is an *axial* permutation if $p^2 = I$ and p is a non-identity permutation.

Lemma 1. *[11] An automorphism of a graph is geometric if and only if it is either an axial permutation or a rotational permutation on the vertex set.*

The ways in which geometric automorphism groups can be combined is not as simple as for automorphism groups. It is trivial to see that the union of two automorphism groups generates an automorphism group; thus there is a single maximal automorphism group. However, a graph can have geometric automorphism groups that are distinct and not combinable; the union of two geometric automorphism groups does not necessarily generate a geometric automorphism group. This is shown by Figure 1(b) and (c). The graph K_4 has a geometric automorphism group of size 8 (Figure 1(b)), and a geometric automorphism group of size 6 (Figure 1(c)). However, these groups do not combine to make a geometric automorphism group; each is maximal in that it cannot be made larger. In fact, Figure 1(b) displays 8 automorphisms, and the maximum size geometric automorphism group of K_4 has size 8.

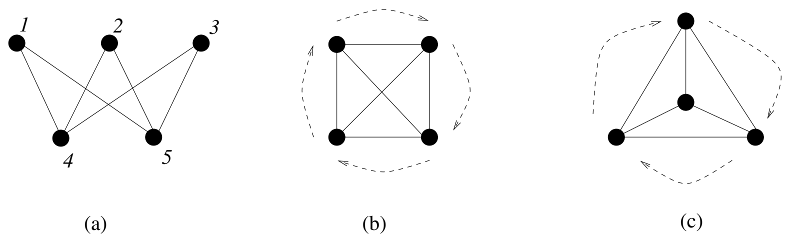


Fig. 1. *Three drawings with different symmetries.*

Our aim is to find drawings with a maximum number of symmetries, and this means finding a maximum size geometric automorphism group. Thus the central problem in drawing graphs symmetrically is as follows.

Geometric Automorphism Problem(GAP)
Input: A graph G.
Output: A maximum size geometric automorphism group of G.

It has been shown [12,15] that GAP is NP-hard.

2.3 Planar Geometric Automorphisms

The concepts above can be extended to planar drawings: a geometric automorphism α of a graph G is *planar* if there is a planar drawing of G which displays α, and an automorphism group P is a *planar geometric automorphism group* if there is a planar drawing which displays P.

Not every geometric automorphism is planar. For example, the rotational automorphism displayed in Figure 1(b) is not planar; the largest planar geometric automorphism group of K_4, displayed in Figure 1, has size 6.

The central problem of this paper is to find a planar geometric subgroup of maximum size.

> *Planar Geometric Automorphism Problem(PGAP)*
> *Input:* A planar graph G.
> *Output:* A planar geometric subgroup P of the automorphism group of G, such that P has maximum size.

Previous research on PGAP has concentrated on subclasses of the class of planar graphs. An algorithm of Manning [15] finds planar geometric automorphisms of a fixed planar embedding (with a fixed outer face) in linear time. For a triconnected graph there are only $O(n)$ planar embeddings; by selecting each outer face in turn, one can solve PGAP for triconnected graphs in quadratic time. It has been solved for trees [13,15] and series-parallel graphs [6]. In this paper we investigate PGAP for planar graphs. The difficulties are mainly in the transformations between connectivity classes.

3 Finding Planar Geometric Automorphisms

In this section we outline tools for finding planar geometric automorphisms in planar graphs. Many details are omitted for this extended abstract; a more complete description appears in [5].

We use connectivity to divide the problem into cases.

1. G is triconnected.
2. G is biconnected.
3. G is one-connected.
4. G is not connected.

For each case, our algorithm takes for input a graph with vertices and edges possibly labeled, and returns a planar geometric automorphism group for the graph, as well as (possibly) some labels for the vertices and edges. Each case relies on the result of the previous case.

The first case can be dealt with by the algorithm of Manning [15]. The second case is the most difficult, and it is described in the Section 3.2. The third case is described in Section 3.1. The fourth case is omited in this extended abstract.

3.1 The One-Connected Case

The *block-cutvertex tree* of a connected graph G [2] has a B-node for each block (biconnected component) of G, and a C-node for each cutvertex of G. There is an edge between a C-node u and a B-node b if and only if u belongs to the corresponding block of b in G. The block-cutvertex tree can be constructed in linear time [17]. A *center* of a tree is a vertex whose maximum distance from a leaf is minimized. Note that in a block-cutvertex tree, there is only one center (since every leaf is a B-node). The center of the block-cutvertex tree may be a block or a cutvertex.

To find geometric automorphisms of a one-connected graph G we need to use the method for biconnected (described in Section 3.2) graphs as a subroutine.

The method proceeds from the leaves of the block-cutvertex tree to the center; we may regard the block-cutvertex tree as rooted at the center. At each stage, all the blocks corresponding to leaves of the block-cutvertex tree are processed. Each cutvertex u of the input graph has three labels. Note that a cutvertex occurs in more than one block and in general it has different labels in different blocks. The three labels are:

– A boolean label u_P^B, which has the value true when u is the cutvertex that joins the block B to its parent in T.
– An isomorphism code u_C^B. This is a set of integers; the number of elements in u_C^B is the number of children of u in T. If u and v are two cutvertices with $u_C^B = v_C^B$ then the subgraph of G graph represented by the subtree under u in T is isomorphic to the subgraph represented by the subtree under v.
– A label u_A^B, which indicates whether the subgraph represented by the subtree under u has an axial planar automorphism which fixes the cutvertex that joins B to its parent. If such an automorphism exists, then this label also indicates the maximum number of edges incident with u which are fixed by the automorphism.

The algorithm begins with the labeling phase, as follows.

1. Construct the block-cutvertex tree T, and for each cutvertex u and each block B, compute u_P^B.
2. For each cutvertex u and each block B, initialize $u_C^B = u_A^B = null$.
3. Repeat until T has one node (the center):
   (a) Compute the labeled automorphism partition of the set S of blocks of G corresponding to leaves of T. Use the automorphism partition to compute u_C^B for each cutvertex u in each block $B \in S$.
   (b) Use the algorithm for biconnected planar graphs (in the next section) to test whether each $B \in S$ has an axial planar geometric automorphism which fixes the cutvertex in B. This gives u_A^B.
   (c) Remove all the leaves of T from T.

After the labeling phase, we process the center of the tree T. If the center is a block, then we call the method for (labeled) biconnected graphs in the next section. If it is a cutvertex, then a simpler algorithm applies; details are omitted for this abstract.

3.2 The Biconnected Case

If the input graph is biconnected, then we break it into triconnected components in a way that is suitable for the task. The method is similar to but more complex than the method of the previous section.

First we review the definition of *3-blocks* or *triconnected components* of a biconnected graph. If G is triconnected, then G itself is the unique triconnected component of G. Otherwise, let u, v be a separation pair of G. We split the edges of G into two disjoint subsets E_1 and E_2, such that $|E_1| > 1$, $|E_2| > 1$, and the subgraphs G_1 and G_2 induced by E_1 and E_2 only have vertices u and v in common. Form the graph G'_1 by adding an edge (called a *virtual edge*) between u and v; similarly form G'_2. We continue the splitting process recursively on G'_1 and G'_2. The process stops when each resulting graph reaches one of three forms: a triconnected simple graph, a set of three multiple edges (triple bond), or a cycle of length three (triangle). The triconnected components of G are obtained from these resulting graphs by merging the triple bonds into maximal sets of multiple edges (bonds), and the triangles into maximal simple cycles (polygons). The triconnected components of G are unique. See [9] for further details.

We can define the *3-block* tree as follows[1]. The vertices of the 3-block tree are the triconnected components of G. The edges of the 3-block tree are the virtual edges, that is, if two triconnected components have a virtual edge in common, then the vertices that represent the two triconnected components in the 3-block tree are joined by an edge that represents the virtual edge. Note that the tree has only one center; the rooted tree rooted at the center is unique [1].

The 3-block tree may be constructed in linear time by adjusting the algorithm of Hopcroft and Tarjan [9]. An example of a biconnected graph and its 3-block tree (from [9]) is in Figure 2.

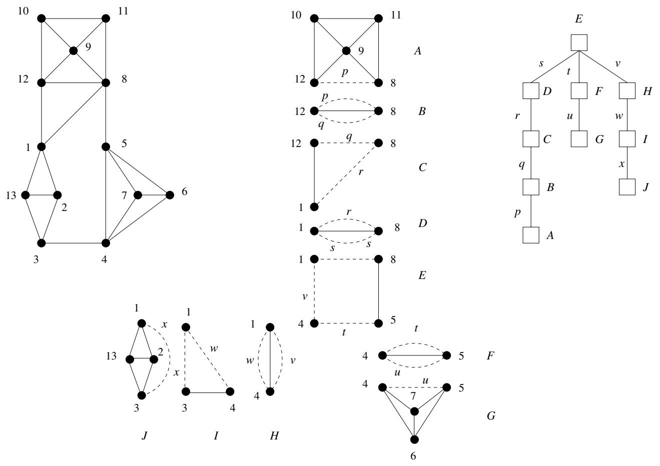


Fig. 2. *Example of 3-Block Tree.*

[1] The *SPQR-tree* defined by di Battista and Tamassia [1] is closely related. There are many variants of the 3-block tree in the literature; the first was defined by Tutte [18].

We use a kind of "reduction" [8], similar to the labeling process for the block-cutvertex tree defined in the previous section. The operation takes a 3-block tree of a biconnected graph G and deletes the triconnected components represented by the leaves. It then adds and labels new edges between the separation pairs[2]. We repeat this process until G becomes a single triconnected component (the center of the 3-block tree). The process is called *tree-shrinking* by Wong [20].

As the triconnected component A is deleted from G, an isomorphism code is attached to the new edge e joining its separation pair in the remaining part of the graph G. In fact, the method assigns a *pair* of isomorphism codes to each triconnected component. This is because the triconnected component has an orientation with respect to its separation pair. Suppose that A and B are two triconnected components which share the separation pair u, v; suppose that, in A, (u, v) has the code (x_A, y_A) and, in B, (u, v) has the code (x_B, y_B). The relationship between A and B is encoded in these pairs as follows.

– If $x_A = x_B$ and $y_A = y_B$ then there is an automorphism of $A \cup B$ which fixes u and v and swaps the other vertices of A with the other vertices of B. (See Figure 3(a).)
– If $x_A = y_B$ and $y_A = x_B$ then there is an automorphism of $A \cup B$ which swaps u and v and swaps the other vertices of A with the other vertices of B. (See Figure 3(b).)
– If $x_A = y_A$ then A has an axial automorphism which swaps u and v. (See Figure 3(c).)

This pair can be computed using the algorithm of Hopcroft and Tarjan for automorphism partition of triconnected components [8].

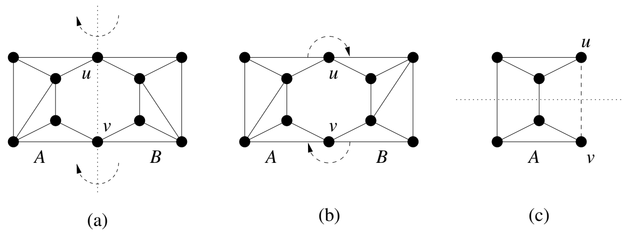


Fig. 3. *Examples for isomorphism codes.*

In addition to the pair of isomorphism codes, we attach further information about the axial automorphisms of a triconnected component. We add this information to the new edge.

To find geometric automorphisms in a planar graph, we need a method to test axial automorphisms about a separation pair[3]. This is illustrated in Figure 4:

[2] Note that multiple edges may occur.
[3] This is related to Manning's *A-class* test.

if B has an axial automorphism, then there are two axial automorphisms of the whole graph. However, if it does not, then the number of axial automorphisms is one. Details of the computation are omitted.

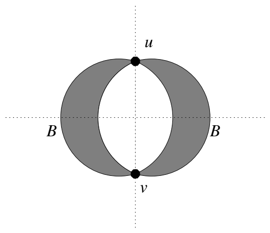


Fig. 4. *Example for testing axial automorphisms.*

3.3 Time Complexity

If G is triconnected then as mentioned previously, Manning's linear time algorithm for finding geometric automorphism in the embedded planar graph [15] can be used for each of the $O(n)$ planar embeddings. Thus it takes $O(n^2)$ time.

When G is composed of one biconnected component, the most expensive operation is finding axial automorphisms of each of the triconnected components. Each test takes time which is quadratic in the size of the triconnected component. Thus in total we use $O(n^2)$ time. A similar argument holds for the one-connected case. Thus our algorithm has $O(n^2)$ time complexity, where n is the number of vertices.

4 Conclusion

In this paper we present an algorithm for finding geometric automorphisms in a planar graph. This algorithm shows that the PGAP problem can be solved in polynomial time, in contrast to the more general GAP problem.

The relatively high complexity $O(n^2)$ is mainly caused by the need to determine geometric automorphisms of triconnected graphs; we just naively adopt Manning's algorithm for each planar embedding, resulting in $O(n^2)$. The complexity of the whole algorithm could be reduced if the dependence on the choice of outer face were not so critical.

Further, we would like to turn our attention to the drawing phase. For triconnected graphs, one can make symmetric drawings (given a geometric automorphism group) subject to a variety of constraints (for example, straight line edges). However, extending these methods seems to be challenging.

References

1. G. Di Battista and R. Tamassia, On-Line Maintenance of Triconnected Components with SPQR-Trees, *Algorithmica* 15, pp. 302-318, 1996. 283
2. J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North Holland, Amsterdam, 1976. 281
3. P. Eades and X. Lin, Spring Algorithms and Symmetry, *Computing and Combinatorics,* Springer Lecture Notes in Computer Science 1276, (Ed. Jiang and Lee), pp. 202-211. 279
4. M. Fontet, Linear Algorithms for Testing Isomorphism of Planar Graphs, *Proceedings Third Colloquium on Automata, Languages, and Programming,* pp. 411-423, 1976. 278, 279
5. S. Hong, Geometric Symmetry of Graphs and their Drawing Algorithms, KOSEF (Korea Science and Engineering Foundation) Report, 1998 (in Korean). 281
6. S. Hong, P. Eades, A. Quigley and S. Lee, Drawing Algorithms for Series-Parallel Digraphs in Two and Three Dimensions, GD98 (Proceedings of the Sixth Symposium on Graph Drawing), Lecture Notes in Computer Science, Springer (to appear). 278, 281
7. J. E. Hopcroft and R. E. Tarjan, A VlogV Algorithm for Isomorphism of Triconnected Planar Graphs, *J. Comp. System Sci.* 7, pp. 323-331, 1971. 278
8. J. E. Hopcroft and R. E. Tarjan, Isomorphism of Planar Graphs, *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, pp. 131-151, 1972. 278, 284
9. J. E. Hopcroft and R. E. Tarjan, Dividing a Graph into Triconnected Components, *SIAM J. on Comput.* 2, pp. 135-158, 1973. 283
10. J. E. Hopcroft and J. K. Wong, Linear Time Algorithm for Isomorphism of Planar Graphs, *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing,* pp. 172-184, 1974. 278, 279
11. X. Lin, *Analysis of Algorithms for Drawing Graphs*, Ph.D. Thesis, University of Queensland, 1992. 279, 280
12. A. Lubiw, Some NP-Complete Problems Similar to Graph Isomorphism, *SIAM Journal on Computing* 10(1):11-21, 1981. 278, 280
13. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Trees, *Congressus Numerantium* 64, pp. 159-169, 1988. 277, 278, 279, 281
14. J. Manning and M. J. Atallah, Fast Detection and Display of Symmetry in Outerplanar Graphs, *Discrete Applied Mathematics* 39, pp. 13-35, 1992. 277, 278, 279
15. J. Manning, *Geometric Symmetry in Graphs*, Ph.D. Thesis, Purdue Univ., 1990. 277, 278, 279, 280, 281, 285
16. R. Mathon, A Note on Graph Isomorphism Counting Problem, *Information Processing Letters* 8, 1979, pp. 131-132. 278
17. R. E. Tarjan, Depth-first Search and Linear Graph Algorithms, *SIAM J. Comput.* 1, pp. 146-160, 1972. 282
18. W. T. Tutte, Graph Theory, Encyclopedia of Mathematics and Its Applications, Vol. 21, *Addison-Wesley,* Reading, MA, 1984. 283
19. H. Wielandt, Finite Permutation Groups, *Academic Press,* 1964. 278
20. J. K. Wong, *Isomorphism Problems Involving Planar Graphs*, Ph.D. Thesis, Cornell Univ., 1975. 284

New Approach for Speeding Up Enumeration Algorithms

Takeaki Uno

Dept. Industrial Engineering and Management, Tokyo Institute of Technology
2-12-1 Oh-okayama, Meguro-ku, Tokyo 152, Japan
uno@me.titech.ac.jp

Abstract. We propose a new approach for speeding up enumeration algorithms. The approach does not relies on data structures deeply, but utilizes some analysis of its computation time. The enumeration algorithms for directed spanning trees, matroid bases, and some bipartite matching problems are speeded up by this approach. For a given graph $G = (V, E)$, the time complexity of the algorithm for directed spanning tree is $\mathrm{O}(\log^2 |V|)$ per a directed spanning tree. For a given matroid $\mathcal{M}$, the algorithm for matroid bases runs in $\mathrm{O}(T/n)$ time per a base. Here n denotes the rank of $\mathcal{M}$, and T denotes the computation time to obtain elementary circuits. Enumeration algorithms for matching problems spend $\mathrm{O}(|V|)$ time per a matching.

1 Introduction

For many graph and geometry objects, enumeration algorithms have been developed. Speeding up is one of important and interesting parts of the studies on enumeration algorithms. There are many algorithms and their improvements, especially for spanning trees and paths [1,2,3,4], although neither generalized technique nor framework is proposed for these improvements. Almost all fast enumeration algorithms are improved by speeding up their iterations with some data structures. Hence if we can not speed up iterations, we may obtain no fast algorithm.

In this paper, we propose a new approach "trimming and balancing" for speeding up enumeration algorithms. Our approach is not based on data structures, hence we can apply it to many enumeration algorithms which have not been improved in the existing studies. Our approach adds two new phases to an enumeration algorithm, which are called the trimming phase and the balancing phase. By adding these, an iteration of a modified algorithm may take much computation time than original one, but the total time complexity of the modified algorithm is often considerably smaller than the original. Some our algorithms with the worst case time complexity of an iteration larger than the original one often attain a better upper bound of the time complexity than the original. The time complexity of a trimming and balancing algorithm is not so easy to analyze. We use a technique to analyze time complexities.

In the next section, we show the framework of the approach and the technique of our analysis. We also show a trimming and balancing algorithm for the enumeration problem of directed spanning trees.

2 Framework of Trimming and Balancing

At the beginning of this section, we explain the idea of the trimming phase and balancing phase. To explain it, we use a binary partition algorithm for the enumeration problem of directed spanning trees whose root is a specified vertex r. A directed spanning tree (denoted by DST) is a spanning tree of a digraph satisfying that no its arc shares its head with the other. The algorithm inputs a digraph, and chooses an not "unnecessary" arc a^* (which is called a partitioning arc). An unnecessary arc is an arc included in all DSTs or no DST. The algorithm divides the problem into two subproblems of enumerating all DSTs including a^*, and all those not including a^*. These subproblems can be solved recursively with the graph obtained by removing all the arcs sharing their heads with a^*, and that obtained by removing a^*. If there is only one DST, then all the arcs are unnecessary. Hence the algorithm stops. The algorithm is known to take $O(|A|)$ time per an iteration, and per an outputted DST. Here we show the details of the algorithm.

```
ALGORITHM: ENUM_DST ($G = (V, A)$ )
Step 1: Find a partitioning arc $a^*$. If there are only unnecessary arcs,
        then output the unique DST and stop.
Step 2: Remove all arcs sharing their heads with $a^*$, and call ENUM_DST($G$).
Step 3: Remove $a^*$ from $G$, and call ENUM_DST ($G$) recursively.
```

The trimming phase removes unnecessary parts from the input. In this algorithm, unnecessary arcs can be removed or contracted, since removals and contraction of these arcs effect no change to the original problem. These removals and contractions reduce the size of the input, thus the reduced input includes many outputs for its size. For example, as we show in the later section, a digraph $G = (V, A)$ including no unnecessary arc ("unnecessary" is characterized in the just above) contains $\Omega(|A|)$ DSTs. By the trimming phase, the computation time of an iteration will be not so large for the number of outputs, hence the total computation time per an output will be small.

The trimming phase decreases the computation time but does not always decrease the computation time of the worst case. Suppose that the algorithm inputs a digraph shown in the Figure 1. As we can see, no arc is unnecessary. Now we suppose that the algorithm choose the arc a as a partitioning arc. Since only one DST includes a, one of the subproblem terminates immediately. To construct the other subproblem, a is removed. By the trimming algorithm, the arc sharing its head with a is contracted. The shape of the obtained graph is same as the original. If all the subproblems occurring in the algorithm choose the end-arc as a partitioning arc like the above, the total computation time will

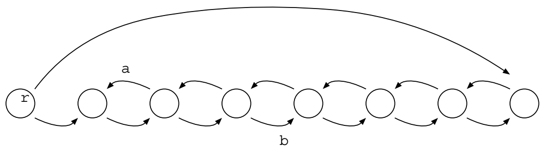


Fig. 1. A digraph for enumeration problem of DSTs

be $O(|A|+(|A|-2)+(|A|-4)+...+4)$. One of the subproblems of them outputs a DST, hence the computation time is $O(|A|)$ per a DST.

Why the worst case running time does not decrease? The answer is that the choosing rule of partitioning arcs is bad. To reduce the time complexity, we have to make a good rule. The balancing phase is added for this reason. It chooses a partitioning arc such that both generated subproblems output not so few DSTs. The subproblems have not so many arcs for the number of outputs after the trimming phase, since we have a lower bound of the number of DSTs for its input size. Thus, if the input includes few DSTs like the above example, then the inputs of both subproblems will be small. Moreover, the depth of the recursion is small in this case, hence the total time complexity will be reduced. As we show in the later section, there is an arc such that both generated subproblems have at least $|A|/4$ arcs after the trimming phase. In the figure 1, the arc b is such an arc. The both generated subproblems by b have about $|A|/2$ arcs after the trimming phase. The shapes of the obtained graphs are also like the figured graph. Hence, by choosing partitioning arcs similarly, the depth of the recursion is $O(\log|A|)$. The total time complexity is also reduced about $O(\log|A|)$ per a DST, if those phases take only $O(|A|)$ time.

In fact, the trimming and balancing algorithm for this problem described in the followings is not so simple like the above. We have to remove some more unnecessary parts from inputs. The trimming phase takes $O(|A|\log|A|)$ time, hence the total computation time is $O(\log^2|A|)$ per a DST.

The trimming and balancing approach decreases the time complexities of enumeration algorithms, but it is not easy to estimate a good upper bound of those time complexities. The above example is simple one. There are some difficult algorithms to analyze their time complexities. In this paper, we also show a technique for analyzing the time complexities of enumeration algorithms. We explain it in the following.

Before explaining our analysis, we introduce a virtual tree called an *enumeration tree* of an enumeration algorithm, which expresses the structure of the recursive calls. For a given enumeration algorithm and its input, let $\mathcal{V}$ be a vertex set whose elements correspond to all recursive calls occurring in the algorithm. We consider an edge set $\mathcal{E}$ on $\mathcal{V}\times\mathcal{V}$ such that each whose edge connects two vertices if and only if a recursive call corresponding to one of the vertices occurs in the other. Since the structure of a recursive algorithm contains no circulation, the graph $\mathcal{T}=(\mathcal{V},\mathcal{E})$ forms a tree. This tree is called an enumeration tree of the algorithm. The root vertex of the tree corresponds to the start of the algorithm. To analyze enumeration algorithms, we show some properties of enumeration trees which satisfy for any input.

To analyze the time complexity of an enumeration algorithm, we consider the following distribution rule on the enumeration tree. Let $\mathcal{T}$ be an enumeration tree, and $D(x)$ be the number of descendants of a vertex x of $\mathcal{T}$. Suppose that $T(x)$ is an upper bound of the time complexity of an iteration x. We also suppose that $\hat{T}$ is an upper bound of $\max_{x\in\mathcal{T}}\{T(x)/D(x)\}$. Our analysis uses a parameter T^*. The distribution is done from a vertex to its children in the top-down manner. For a vertex x of the enumeration tree, let $T_p(x)$ be the computation time distributed by the parent of x to x. We distribute $T_p(x)+T(x)-T^*$ of the computation time to its children such that each child receives the computation time proportional to the number of its descendants. We distribute the computation time of each child recursively.

By this distribution rule, some vertices may receive much computation time. Thus we define excess vertices for a specified positive constant $\alpha > 1$, and stop the distribution on the excess vertices. A vertex x is called *excess* if $T_p(x) + T(x) > \alpha\hat{T}D(x)$. The children of an excess vertex receive no computation time from their parent. The distribution rule is also applied to the descendants of excess vertices. By this new rule, $T_p(x)$ is bounded by $\alpha\hat{T}D(x)$ for any vertex x, since the computation time distributed from a parent to its child is proportional to the number of descendants of the child.

After the distribution, no vertex except excess vertices has more than $O(\hat{T} + T^*)$ on it. Next, we distribute the computation time on each excess vertex x to all its descendants uniformly. Since the excess time $T_p(x) + T(x) - T^*$ is bounded by $(\alpha + 1)\hat{T}D(x)$, each descendant receives at most $(\alpha + 1)\hat{T}$ time from an excess ancestor. Let X^* be an upper bound of the maximum number of the excess vertices on a path from the root to a leaf. By using X^* we obtain an upper bound $O(T^* + \hat{T}X^*)$ of the time complexity per an iteration. From these facts, we obtain the following theorem.

Theorem 1. *An enumeration algorithm terminates in* $O(T^* + \hat{T}X^*)$ *time per an iteration.* □

Our analysis requires $\hat{T}$ and X^*. To obtain a good upper bound of the time complexity, we have to set X^* and $\hat{T}$ to sufficiently good values. As a candidate of $\hat{T}$, we can utilize $\max_{x\in\mathcal{T}} T(x)/\bar{D}(x)$ where $\bar{D}(x)$ is a lower bound of $D(x)$. In the enumeration tree, it is hard to identify excess vertices, although we can obtain an efficient upper bound X^*. Let x and y be excess vertices such that y is an ancestor of x and no other excess vertex is in the path P_{yx} from y to x in the enumeration tree. Note that P_{yx} has at least one internal vertex.

Lemma 1. *At least one vertex* w *of* $P_{yx} \setminus y$ *satisfies the condition that* $T(w)$ *is larger than the sum of* $\frac{\alpha}{\alpha+1}T(u)$ *over all children* u *of* w*.*

Proof. If $P_{yx} \setminus y$ includes no such vertex, then all vertices w of $P_{yx} \setminus y$ satisfy the condition that $T_p(w) \le \alpha\hat{T}D(w)$. It contradicts to the assumption of the statement. We prove it by induction. Any child of y satisfies the condition since y is an

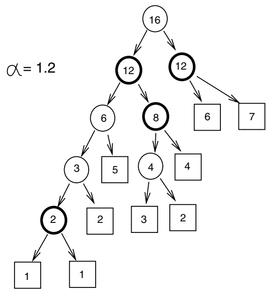


Fig. 2. The enumeration tree and computation time on each vertex: The vertices satisfying the condition of Lemma 1 are drawn by emphasized circles, and all leaves are drawn by rectangles. In this tree, we can set $\hat{T}$ to 7, and X^* to 2.

excess vertex. Suppose that a vertex w of $P_{yx} \backslash y$ holds $T_p(w') \leq \alpha T(w')$, where w' is the parent of w. Then $T_p(w) \leq (\alpha+1)T(w')D(w)/D(w')$. From the assumption, $T(w')$ is not greater than the sum of $\frac{\alpha}{\alpha+1}\hat{T}D(u)$ over all children u of w'. Since the sum of $D(u)$ is not greater than $D(w')$, we have $T(w') \leq \frac{\alpha}{\alpha+1}\hat{T}D(w')$. Therefore we have $T_p(w) \leq \frac{\alpha(\alpha+1)}{\alpha+1}\hat{T}D(w')D(w)/D(w') = \alpha\hat{T}D(w)$. □

From this lemma, we can obtain X^* by estimating an upper bound of the number of vertices satisfying this condition in any path from the root to a leaf. Similarly, we can obtain the following corollary.

Corollary 1 *If $\hat{T} = \max_{x \in \mathcal{T}}\{T(x)/\bar{D}(x)\}$, a vertex w of $P_{yx} \setminus y$ satisfies that $\bar{D}(w)$ is larger than the sum of $\frac{\alpha}{\alpha+1}\bar{D}(u)$ over all children u of w.* □

These conditions can be easily checked, and are often sufficient to analyze. In the following sections, we describe one of them, that for DSTs. To see the algorithms for perfect matchings, refer [5].

3 Enumerating Directed Spanning Trees

In this section, we consider an enumeration algorithm for DSTs. The simple binary partition algorithm explained in the above section is proposed by H.N.Gabow and E.W.Myers [1] in 1978. Our algorithm is obtained by adding a trimming phase and a balancing phase to this algorithm. We describe these algorithms in the following sections.

3.1 A Trimming Algorithm

Our trimming algorithm removes unnecessary parts of input. Firstly, the algorithm removes all multiple arcs since at most one of multiple arcs can be included in a DST. Next we see characterizations for unnecessary arcs which are included in all DSTs or no DST.

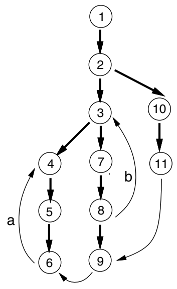


Fig. 3. Back arc a is included in a DST since $i(h(6,6)) = 2$. Back arc b is included in no DST since $i(h(7,7)) = 3$ and $i(h(8,8)) = 7$.

Property 1 *For an arc (u,v), there is a DST including it if and only if there is a simple dipath from r to u not including v.* □

Property 2 *For an arc (u,v), there is a DST not including it if and only if there is a simple dipath from r to v not including (u,v).* □

From Property 2, if all arcs not satisfying the former condition are removed, then an arc is not included in a DST if and only if it shares its head with the other arc. Let T be a depth-first search tree of G with the root r. For a non-back arc a of T, the dipath from r to the tail of a in T does not include the head of a. Thus the former condition holds for any non-back arc. In the following, we show a way for checking the condition for back arcs.

Let us put indices $i(v)$ to all vertices v in the order of visiting of the depth-first search. We denote the unique path in T from a vertex u to its descendant v by P_{uv}. For an index i, vertices v and u, we call a dipath from u to v an *i-bypass* if all its internal vertices have indices larger than or equal to i. For a vertex v and an index $i \leq i(v)$, let $h(v,i)$ be the minimum index vertex among vertices satisfying that there are some i-bypasses from the vertices to v. Since any dipath to v from a vertex with an index smaller than v includes a common ancestor of them, $h(v,i)$ is an ancestor of v. Note that an ancestor has an index smaller than any its descendant. By using these notations, we state the following lemmas (see Figure 3).

Lemma 2. *For a back arc (u,v), there is a dipath from r to u not including v if and only if $i(h(w,i(w))) < i(v)$ holds for a vertex w in $P_{vu} \setminus v$.*

Proof. The "if" part of the lemma is obviously. Suppose that there is a simple dipath P from r to u not including v. Then P includes some vertices of $P_{vu} \setminus v$. Let w be the first vertex of them to appear in the path. The subpath of P from r to w includes some ancestors of v. Let w' be the last vertex of them to appear in the subpath. Note that no vertex of $P_{w'w}$ is an internal vertex of the subpath from w' to w of P. Moreover, the subpath includes no vertex u with $i(u) < i(v)$ since any path from u to v includes some common ancestors of u and v, since T is a depth-first search tree. It contradicts the choosing way of w'. □

From this lemma, we can identify unnecessary arcs by using h. Our algorithm firstly obtains $h(v, i(v))$ for the vertex v with $i(v) = n$. In each iteration, we decrease i one by one, and obtain $h(v, i)$ from $h(v, i+1)$ for all v with $i(v) \geq i$. The updating method of h is based on the following properties.

Lemma 3. *Suppose that a vertex u has the index $i-1$.*
(1) $h(u, i-1)$ is the minimum index vertex among all v and $h(v, i)$ satisfying that there are arcs (v, u).
(2) If $h(v, i) \neq h(v, i-1)$ holds for a vertex v, then v is a descendant of u and holds $h(v, i-1) = h(u, i-1)$.

Proof. (1) Let P be an $i-1$-bypass from $h(u, i-1)$ to u. If P is not an arc from $h(u, i-1)$ to u, the vertex v next to u satisfies $h(v, i) = h(u, i-1)$. Thus the condition holds. (2) If $h(v, i-1) \neq h(v, i)$, u is included in any $i-1$-bypass from $h(v, i-1)$ to v. Hence $h(v, i-1) = h(u, i-1)$. Since (a) a dipath from u to a vertex with an index larger than $i-1$ includes a common ancestor of them, and (b) any ancestor of u has an index smaller than $i-1$, v is a descendant of u. Therefore $h(v, i-1) \neq h(v, i)$ holds only for descendants of u. □

From the lemma, we can see that $i(h(v, j)) \leq i(h(v, i))$ for any $j < i$.

Lemma 4. *Let u be a vertex satisfying that $i(u) > i$, and v be a descendant of u. If we have $h(u, i) = h(v, i)$, then $h(u, j) = h(v, j)$ holds for any $j < i$.*

Proof. Since u is an ancestor of v, $i(h(u, j)) \geq i(h(v, j))$ holds for any j. Suppose that an index j satisfies $i(h(u, j)) > i(h(v, j))$. We assume that j is the maximum index among indices satisfying the condition. From this assumption, there is a j-bypass from $h(v, j)$ to v not including u. Note that the bypass includes the vertex v' whose index is j. Since T is a depth-first search tree, any dipath from v' to v includes some common ancestors of v and v'. These ancestors are also ancestors of u, thus we can obtain a j-bypass from $h(v, j)$ to u by merging the j-bypass from $h(v, j)$ to v and $P_{v'u}$. It contradicts the assumption. □

From the lemma, if we have $h(u, i) = h(v, i)$ for u and its child v, the equation holds for all $j < i$. Hence, in the graph obtained by contracting u and v, h is preserved. Thus we contract them if such a vertex and a child exist. For a vertex u, if a descendant v of u satisfies that $i(h(v, i(u)+1)) > i(h(v, i(u)))$, the child v' of u included in the dipath from u to v satisfies the condition $i(h(v', i(u)+1)) > i(h(u, i(u)))$. Hence v' and u satisfy that $i(h(v', i(u))) = i(h(u, i(u)))$. Therefore, in each iteration with the index i, we find the child v' of the vertex u with $i(u) = i$ which maximizes $i(h(v', i))$ among all children of u. If v' satisfies $i(h(v', i)) < i(h(u, i))$, all descendants v of u satisfies $i(h(v, i)) < i(h(u, i))$. Otherwise, we have that $h(v', i) = h(u, i)$, hence we contract u and v'. We do this operation until there is no child v satisfying $i(h(v, i)) = i(h(u, i))$. After contracting all these vertices, no descendant v of u satisfies that $h(u, i) \leq h(v, i+1)$. Therefore no vertex v satisfies $h(v, i) \neq h(v, i+1)$ in the contracted graph. Hence we can update all h by this contracting operation. Since using some binary trees, the total time complexity of the trimming algorithm is $O(|A| \log |V|)$.

3.2 A Balancing Algorithm

For an arc a, let G' be the graph obtained by deleting all arcs sharing their heads with a except for a. Under the condition that G is a trimmed graph, the following lemma holds. Suppose that a dipath from a vertex to an arc is a simple dipath from the vertex to the head of the arc which includes the arc.

Lemma 5. *Let P be a dipath from r to a. If an arc e is included in all DSTs or no DST of G', then the head of e is on P.*

Proof. We state the lemma by proving its contraposition. For an arc e whose head is not on P, there exists a dipath Q in G from r to e, because of Property 1. If Q includes no arc of $G \setminus G'$, then it is also included in G'. Thus e is included in some DSTs of G'. Otherwise, we consider the subgraph given by $Q \cup P \cap G'$. The subgraph includes a simple dipath from r to e, since the head of e is not on P. Thus, only arcs which have their heads on P may be included in no DST of G'.

Since G is trimmed, an arc e whose head is not on P shares its head with some other arcs. The arcs are also included in some DSTs of G'. Hence they are also not included in a DST of G'. Therefore any arc included in all DSTs of G' or no DST of G' has its head on P. □

Let a' be an arc sharing its head with a, and P' be a dipath from r to a'.

Lemma 6. *If an arc e is included in all DSTs or no DST of $G \setminus a$, then its head is on P'. In the case that at least two arcs share their heads with a, for any arc e of the arcs, there are both DSTs including e and not including e.*

Proof. $G \setminus a$ includes the graph obtained by deleting all arcs sharing their heads with a' except for a'. Thus, for any arc whose head is not on P', there are both DSTs including the arc and those not including the arc in $G \setminus a$ from Lemma 5. Therefore the first assertion is proved.

For any arc of $G \setminus a$ sharing its head with a', a dipath from r to it does not include a. Hence, from Property 1, the arc is included in some DSTs of G'. Except for the case that only two arcs a and a' have their heads on v, there are some DSTs not including the arc in $G \setminus a$. □

By using these lemmas, we select a partitioning arc as follows. Let the weight of a dipath be the number of arcs whose heads are on the path. The weight of a dipath P is denoted by $w(P)$. In the balancing phase, we will find an arc a^* satisfying the conditions that (a) the weight of a dipath from r to a^* is at most $3|A|/4$, and (b) for an arc a' sharing its head with a^*, the weight of a dipath from r to the tail of a' is at most $|A|/2$. If an arc a^* satisfies these conditions, $G \setminus a^*$ includes at most $|A|/2 + 2$ arcs which are included in all DSTs or no DST. Under the condition that $|A| \geq 8$, $|A|/2 + 2 \leq 3|A|/4$. The graph obtained by removing all arcs sharing whose heads with a^* also includes at most $3|A|/4$ such arcs. We consider the method for finding such an arc a^*.

Let T be a DST of G. To select an arc, we consider the following three cases. (1) If there is a non-back arc (u, v) of T such that $w(P_{ru}) \leq |A|/2$

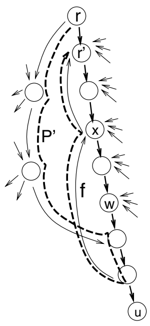


Fig. 4. Vertices r, r', u, x and w. The paths are P_{ru} and P'.

and $w(P_{rv}) \leq |A|/2$, then the arc of T whose head is v satisfies the above conditions. (2) In the other case, let u be the vertex maximizing $w(P_{ru})$ among all vertices. We denote the vertex next to r in P_{ru} by r'. Let P' be a simple dipath from r to r' not including the arc (r, r'). P' always exists since G is a trimmed graph. Let w be the vertex minimizing $w(P_{rw})$ among all vertices v of P_{ru} satisfying that $w(P_{rv}) > |A|/2$. We suppose that x denotes the first vertex to appear in P' among vertices v of P_{ru} with $w(P_{rv}) \leq |A|/2$. Since any vertex except r is the head of at least two arcs, at least $2|V| - 4$ arcs of G have their heads not on w. On the other hand, at most $|V| - 1$ arcs have their heads on w, thus the number of arcs whose heads are w does not exceed $|A|/2$. Therefore the weight of r' is at most $|A|/2$, and x always exists in P'. Let f be the arc of P' whose head is x. We show an example of these vertices and arcs in Figure 3. If the subpath P'' of P' from r to f satisfies $w(P'') \leq |A|/2$, then the arc of T sharing its head with f satisfies the conditions (a) and (b).

(3) If f does not satisfies these conditions, $w(P'') > |A|/2$. From the definition of x, the vertices included in both P'' and P_{rw} are at most r and w. Hence P'' includes w since more than $|A|/2$ arcs have their heads on P_{rw}. Suppose that P_1 denotes the path with the smaller weight among P_{uw} and the subpath of P'' from r to w. We also denote the other path by P_2. Let d denote the number of arcs whose heads are w. Since $P_1 \cap P_2 = \{r, w\}$, we have $w(P_1) \leq (|A| - d)/2 + d = |A|/2 + d/2 \leq 3|A|/4$, and $w(P_2 \setminus w) \leq |A|/2$.

From the above observations, there is an arc satisfying the above two conditions in any trimmed digraph. To find such an arc, what we have to do is only to find a DST and some dipaths. They can be done in $O(|A| + |V|)$ time.

By utilizing these algorithms, we obtain a trimming and balancing algorithm. The algorithm takes $O(|A| \log |V|)$ time for an iteration in the worst case. In the following subsection, we bound the time complexity more tightly by using the distribution of computation time.

3.3 Bounding the Total Time Complexity

To estimate an amortized time complexity of our algorithm, we firstly estimate a lower bound of the number of descendants of a vertex of the enumeration tree by the following lemma. Let G_x denote the graph which an iteration x inputs.

Lemma 7. *If any arc of G is included in a DST and not included in the other DST, G includes at least $|A|/2$ distinct DSTs.*

Proof. Let T be a DST of G. From Property 1, for any non-tree arc a of T, there is a simple dipath P from r to a. Hence, for any non-tree arc, we can construct its own DST T' by adding P to T and deleting some arcs. Since there are at least $|A|/2$ non-tree arcs in G, G includes at least $|A|/2$ DSTs. □

From the lemma, we obtain a lower bound $\bar{D}(x) = |E(G_x)|/2$ of $D(x)$. Here $D(x)$ is the number of descendants of a vertex x of the enumeration tree, and $E(G_x)$ denotes the arc set of G_x. The computation time on a vertex x is $O(|E(G_x)| \log |V|)$ time, hence we set $\hat{T} = O(\log |V|)$, and $T^* = O(\log |V|)$.

From Corollary 1, we can bound the number of excess vertices by the number of vertices satisfying that $\bar{D}(x) > \frac{\alpha}{\alpha+1}(\bar{D}(x_1) + \bar{D}(x_2))$ where x_1 and x_2 are the children of x. If the condition holds, we have that $|E(G_x)|/2 > \frac{\alpha}{\alpha+1}(|E(G_{x_1})|/2 + |E(G_{x_2})|/2)$, thus $|E(G_{x_i})| \le \frac{\alpha+1}{\alpha}|E(G_x)| - |E(G_x)|/4$ for each child x_i. By setting $\alpha = 8$, $|E(G_{x_i})| \le (7/8)|E(G_x)|$. Therefore any path of the enumeration tree from the root to a leaf has at most $\log_{8/7} |A|$ excess vertices. We obtain an upper bound $X^* = \log_{8/7} |A|$. From the Theorem 1, we have the following theorem.

Theorem 2. *All DSTs in a digraph can be enumerated in $O(|A| \log |V| + |V| + N \log^2 |V|)$ time and $O(|A| + |V|)$ space where N is the number of DSTs.* □

Acknowledgments

We greatly thank to Associate Professor Akihisa Tamura of University of Electro-Communications and all the members of the research group "myogadani-club" for their kindly advise.

References

1. H.N.Gabow and E.W.Myers, "Finding All Spanning Trees of Directed and Undirected graphs," SIAM J.Comp.**7**, pp.280-287 (1978). 287, 291
2. H.N.Kapoor and H.Ramesh,"Algorithms for Generating All Spanning Trees of Undirected, Directed and Weighted Graphs," LNCS.**519**, Springer-Verlag, pp.461-472 (1992). 287
3. A.Shioura, A.Tamura and T.Uno, "An Optimal Algorithm for Scanning All Spanning Trees of Undirected graphs, " SIAM J.Comp.**26**, pp.678-692 (1997). 287
4. T.Uno, "An Algorithm for Enumerating All Directed Spanning Trees in a Directed graph," LNCS **1178**, Springer-Verlag, pp.166-173 (1996). 287
5. T.Uno, "Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs," LNCS **1350**, Springer-Verlag, pp.92-101 (1997). 291

Hamiltonian Decomposition of Recursive Circulants[⋆]

Jung-Heum Park

School of Computer Science and Engineering
Catholic University of Korea
Yokkok 2-dong 43-1, Wonmi-gu, Puchon 420-743, Republic of Korea
jhpark@tcs.cuk.ac.kr

Abstract. We show that recursive circulant $G(cd^m, d)$ is hamiltonian decomposable. Recursive circulant is a graph proposed for an interconnection structure of multicomputer networks in [8]. The result is not only a partial answer to the problem posed by Alspach that every connected Cayley graph over an abelian group is hamiltonian decomposable, but also an extension of Micheneau's that recursive circulant $G(2^m, 4)$ is hamiltonian decomposable.

1 Introduction

We say a graph G is *hamiltonian decomposable* if either the degree of G is $2k$ and the edges of G can be partitioned into k hamiltonian cycles, or the degree of G is $2k + 1$ and the edges of G can be partitioned into k hamiltonian cycles and a 1-factor, where a 1-factor of a graph is a 1-regular spanning subgraph. If G is hamiltonian decomposable then G is loopless, connected, and regular.

It is necessary for a graph G to have a hamiltonian decomposition that G has a hamiltonian cycle. However, the condition is by far not sufficient. Many authors have been dealing with sufficient conditions for the existence of a decomposition of a graph into hamiltonian cycles. But still, the current status of the matter lies, for the most part, in the sphere of problems and conjectures.

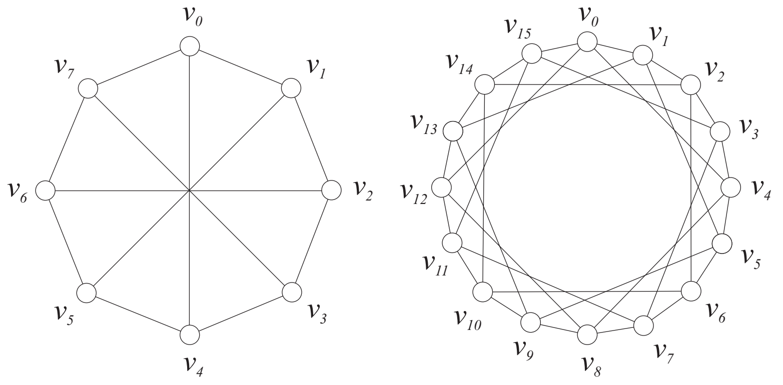
A survey on hamiltonian decomposition of graphs is provided in [3,4]. The complete graph K_n with odd (resp. even) number n of vertices is decomposable into hamiltonian cycles (resp. paths). The complete k-partite graph $K(n_1, n_2, \cdots, n_k)$ is hamiltonian decomposable if and only if $n_1 = n_2 = \cdots = n_k$. The following problem on hamiltonian decomposability of Cayley graphs is posed by Alspach[1].

Problem 1 *Does every connected Cayley graph over an abelian group have a hamiltonian decomposition?*

[⋆] This work was partially supported by the Catholic University of Korea under project no. 19980057.

Much of the focus of research has been directed towards proving special cases of the problem. The answer is yes when the degree of the graph is five or less. The product of any number of cycles $C_{n_1} \times C_{n_2} \times \cdots \times C_{n_k}$ is hamiltonian decomposable. The product of cycles is isomorphic to the Cayley graph of the corresponding cyclic groups with the standard generating set. The m-cube Q_m, the Cayley graph of the product of m copies of a cyclic group Z_2, is hamiltonian decomposable.

Recursive circulant is a graph proposed for an interconnection structure of multicomputer networks in [8]. The recursive circulant $G(N, d)$, $d \geq 2$, is defined as follows: the vertex set $V = \{v_0, v_1, v_2, \cdots, v_{N-1}\}$, and the edge set $E = \{(v_i, v_j) \mid \text{there exists } k,\ 0 \leq k \leq \lceil \log_d N \rceil - 1, \text{ such that } i + d^k \equiv j \pmod{N}\}$. Here each d^k is called a *jump*, and the *size* of the edge (v_i, v_j) is d^k. $G(N, d)$ also can be defined as a circulant graph with N vertices and jumps of powers of d, $d^0, d^1, \cdots, d^{\lceil \log_d N \rceil - 1}$. Examples of $G(N, d)$ are shown in Fig. 1.



(a) $G(8, 4)$ (b) $G(16, 4)$

Fig. 1. Examples of $G(N, d)$

Recursive circulant is a Cayley graph over an abelian group, in more precise words, the Cayley graph of the cyclic group Z_N with the generating set $\{d^0, d^1, \cdots, d^{\lceil \log_d N \rceil - 1}\}$. This paper is concerned with hamiltonian decomposability of recursive circulants. Micheneau shows that recursive circulant $G(N, d)$ with $N = 2^m$ and $d = 4$ is hamiltonian decomposable[7]. We shall prove that $G(N, d)$ has a hamiltonian decomposition when $N = cd^m$, $1 \leq c < d$ for some integer c and d. The result is an extension of Micheneau's as well as a progress on Alspach's problem.

From now on, all arithmetics are done modulo cd^m using the appropriate residues. Graph theoretic terms not defined here can be found in [2]. This paper is organized as follows. We give some preliminaries and related works in Section 2, and prove the main theorem in Section 3 that $G(cd^m, d)$ is hamiltonian decomposable. Finally, concluding remarks are given in Section 4.

2 Preliminaries and Related Works

Let us consider the classes of graphs containing recursive circulants. Recursive circulant $G(N, d)$ is a circulant graph. A circulant graph is defined as a Cayley graph over a cyclic group. Every Cayley graph over a general group is vertex symmetric, and thus regular. These inclusion relationships are shown in Fig. 2 (a). Depending on restriction to N and d, the recursive circulants also have inclusion relationships shown in Fig. 2 (b).

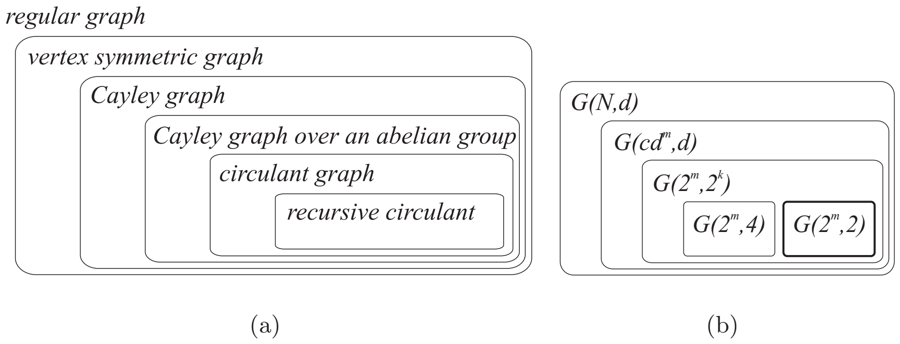


Fig. 2. Graph classes

Hamiltonian decomposition is one of the most interesting strong hamiltonicity, that is, a hamiltonian property which implies the existence of a hamiltonian cycle. Hamiltonian connectedness is also a strong hamiltonian property. A graph is *hamiltonian connected* if there is a hamiltonian path joining every pair of vertices in the graph.

When one is trying to prove that every graph in a class has a certain hamiltonian property and fails to do so, then typically two approaches are taken. One is to restrict the class of graphs and prove that the property holds over the restricted class, while the other is to restrict the property and prove that every graph in the class satisfies the restricted property. Many researches on hamiltonicity of graphs in the literature take these approaches. Some well-known conjectures and impressive properties on hamiltonicity of graphs in the classes that we have interest are shown below.

Conjecture 2 *(a) Every connected vertex symmetric graph has a hamiltonian path[Lovasz].*
(b) Every connected Cayley graph with three or more vertices has a hamiltonian cycle[Chen].
(c) Every $2k$-regular connected Cayley graph on a finite abelian group is hamiltonian decomposable[Alspach].

Theorem 3 *(a) Every connected Cayley graph over an abelian group is hamiltonian connected[5].*
(b) Recursive circulant $G(2^m, 4)$ is hamiltonian decomposable[7].

Recursive circulant $G(N, d)$ with three or more vertices has a hamiltonian cycle. Obviously, the set of edges of size 1 in $G(N, d)$ form a hamiltonian cycle. Theorem 3 (a) shows that $G(N, d)$ has a strong hamiltonicity of hamiltonian connectedness.

Recursive circulant $G(N, d)$ has a recursive structure when $N = cd^m$,$1 \le c < d$. In other words, $G(cd^m, d)$ can be defined recursively by utilizing the following property.

Property 4 *Let V_i be a subset of vertices in $G(cd^m, d)$ such that $V_i = \{v_j \mid j \equiv i \pmod d\}$, $m \ge 1$. For $0 \le i \le d-1$, the subgraph of $G(cd^m, d)$ induced by V_i is isomorphic to $G(cd^{m-1}, d)$.*

$G(cd^m, d)$, $m \ge 1$, can be constructed recursively on d copies of $G(cd^{m-1}, d)$ as follows. Let $G_i(V_i, E_i)$, $0 \le i \le d-1$, be a copy of $G(cd^{m-1}, d)$. We assume that $V_i = \{v_0^i, v_1^i, \cdots, v_{cd^{m-1}-1}^i\}$, and G_i is isomorphic to $G(cd^{m-1}, d)$ with the isomorphism mapping v_j^i to v_j. We relabel v_j^i by v_{jd+i}. The vertex set V of $G(cd^m, d)$ is $\bigcup_{0 \le i \le d-1} V_i$, and the edge set E is $\bigcup_{0 \le i \le d-1} E_i \cup X$, where $X = \{(v_j, v_{j'}) \mid j + 1 \equiv j' \pmod{cd^m}\}$. The construction of $G(32, 4)$ on four copies of $G(8, 4)$ is illustrated in Fig. 3. Every edge in X is of size 1, and X forms a hamiltonian cycle, called the *fundamental hamiltonian cycle*, in $G(cd^m, d)$.

We denote by δ_m the degree of $G(cd^m, d)$. The degree of a graph is the minimum degree over all vertices in the graph. δ_m is greater than δ_{m-1} by two, that is, $\delta_m = \delta_{m-1} + 2$, $m \ge 1$. δ_m in a closed-form is shown below.

$$\delta_m = \begin{cases} 2m - 1 & \text{if } c = 1 \text{ and } d = 2; \\ 2m & \text{if } c = 1 \text{ and } d \ne 2; \\ 2m + 1 & \text{if } c = 2; \\ 2m + 2 & \text{if } c > 2. \end{cases}$$

3 Hamiltonian Decomposition of $G(cd^m, d)$

In this section, we prove the following main theorem by mathematical induction on degree of recursive circulant $G(cd^m, d)$. We have two cases depending on the size of d.

Theorem 5 *Recursive circulant $G(cd^m, d)$ is hamiltonian decomposable.*

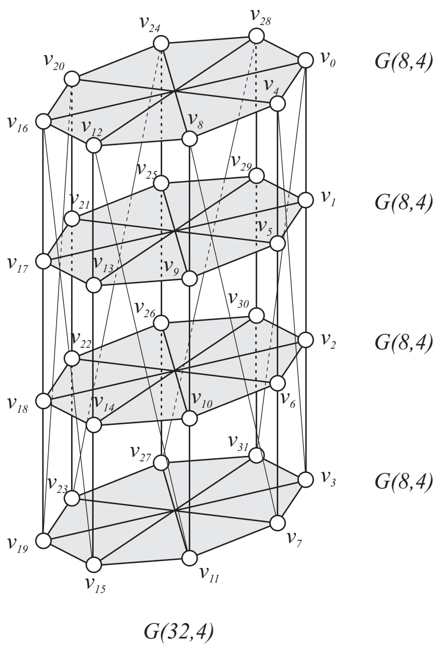


Fig. 3. Recursive structure of $G(32, 4)$

3.1 Case of $d \geq 4$

We show that $G(cd^m, d)$ has a hamiltonian decomposition such that hamiltonian cycles in the decomposition satisfy the following two conditions C1.1 and C1.2.

C1.1 For every hamiltonian cycle C_j in the decomposition, there exists an index k_j such that C_j passes through two adjacent edges (v_{k_j}, v_{k_j+1}) and (v_{k_j+1}, v_{k_j+2}) of size 1.

C1.2 For any pair C_j and $C_{j'}$ of hamiltonian cycles in the decomposition, $\{v_{k_j}, v_{k_j+1}, v_{k_j+2}\} \cap \{v_{k_{j'}}, v_{k_{j'}+1}, v_{k_{j'}+2}\} = \emptyset$.

Let us first consider a hamiltonian decomposition of $G(cd^m, d)$ with small degree. $G(cd^m, d)$ with degree two is the fundamental hamiltonian cycle itself. When the degree of $G(cd^m, d)$ is three, $G(cd^m, d)$ consists of the fundamental hamiltonian cycle and a 1-factor. In both cases, the natural decomposition leads to a hamiltonian decomposition satisfying the above two conditions.

Now we try to find a hamiltonian decomposition of $G(cd^m, d)$ by using the fact that $G(cd^{m-1}, d)$ has a hamiltonian decomposition satisfying the two conditions. The degree of $G(cd^m, d)$ is greater than that of $G(cd^{m-1}, d)$ by two. Note that $G(cd^m, d)$ has a recursive structure, that is, $G(cd^m, d)$ is constructed on d copies of $G(cd^{m-1}, d)$. We let G_i denote a copy of $G(cd^{m-1}, d)$, and assume that the vertices of G_i is $\{v_0^i, v_1^i, \cdots, v_{cd^{m-1}-1}^i\}$. The vertex v_j^i is relabeled by v_{jd+i}. Hereafter, two labels for the vertex are used interchangeably.

We denote by X the fundamental hamiltonian cycle in $G(cd^m, d)$. Let C_j^i denote a hamiltonian cycle in the hamiltonian decomposition of G_i. C_j^i passes through the edges $(v_{k_j}^i, v_{k_j+1}^i)$ and $(v_{k_j+1}^i, v_{k_j+2}^i)$ for some k_j by condition C1.1.

The basic idea of the proof is that we merge d different cycles C_j^i, $0 \leq i < d$, into one hamiltonian cycle C_j in $G(cd^m, d)$ by exchanging some edges of C_j^i's with edges in X while we keep X being a hamiltonian cycle. We merge another d different cycles $C_{j'}^i$ into another hamiltonian cycle $C_{j'}$ by these edge exchange operations. We repeat this until a hamiltonian decomposition of $G(cd^m, d)$ is obtained.

We represent C_j^i and X as a sequence of vertices. It is convenient to represent explicitly only the vertices concerned with edge exchange operations as follows. Here P^i is a path from $v_{k_j+2}^i$ to $v_{k_j}^i$ (excluding the start and end vertices) passing through all the vertices in G_i except $v_{k_j}^i$, $v_{k_j+1}^i$, and $v_{k_j+2}^i$. P is a path from $v_{k_j+2}^{d-1}$ to $v_{k_j}^0$ in $G(cd^m, d)$ passing through all the other vertices not represented explicitly. In other words, P passes through the vertices $\{v_j^i \,|\, 0 \leq i < d, 0 \leq j < cd^{m-1}, j \neq k_j, k_j + 1, k_j + 2\}$ in some order.

$$C_j^i = v_{k_j}^i, v_{k_j+1}^i, v_{k_j+2}^i, P^i$$
$$X = v_{k_j}^0, v_{k_j}^1, \cdots, v_{k_j}^{d-1}, v_{k_j+1}^0, v_{k_j+1}^1, \cdots, v_{k_j+1}^{d-1}, v_{k_j+2}^0, v_{k_j+2}^1, \cdots, v_{k_j+2}^{d-1}, P$$

The edge exchange operation depends on the parity of d.

Case 1 even d

d hamiltonian cycles C_j^i's are merged into a hamiltonian cycle C_j in $G(cd^m, d)$ as follows(See Fig. 4 (a)). Note that X still remains a hamiltonian cycle in $G(cd^m, d)$.

$$\begin{gathered} C_j = v_{k_j}^0, v_{k_j}^1, P^1, v_{k_j+2}^1, \cdots, v_{k_j+2}^{d-4}, P^{d-4}, v_{k_j}^{d-4}, v_{k_j}^{d-3}, P^{d-3}, v_{k_j+2}^{d-3}, v_{k_j+2}^{d-2}, \\ P^{d-2}, v_{k_j}^{d-2}, v_{k_j+1}^{d-2}, v_{k_j+1}^{d-3}, v_{k_j+1}^{d-4}, \cdots, v_{k_j+1}^1, v_{k_j+1}^0, v_{k_j}^{d-1}, P^{d-1}, \\ v_{k_j+2}^{d-1}, v_{k_j+1}^{d-1}, v_{k_j+2}^0, P^0 \\ X = v_{k_j}^0, v_{k_j+1}^0, v_{k_j+2}^0, v_{k_j+2}^1, v_{k_j+1}^1, v_{k_j}^1, \cdots, v_{k_j}^{d-4}, v_{k_j+1}^{d-4}, v_{k_j+2}^{d-4}, v_{k_j+2}^{d-3}, \\ v_{k_j+1}^{d-3}, v_{k_j}^{d-3}, v_{k_j}^{d-2}, v_{k_j}^{d-1}, v_{k_j+1}^{d-1}, v_{k_j+1}^{d-2}, v_{k_j+2}^{d-2}, v_{k_j+2}^{d-1}, P \end{gathered}$$

The edge exchange operations can be performed independently since the hamiltonian decomposition of G_i always satisfies the condition C1.2. Thus every hamiltonian cycle in G_i can be merged into a hamiltonian cycle in $G(cd^m, d)$. If G_i has a 1-factor in the hamiltonian decomposition, G_i has odd degree and even number of vertices. The union of 1-factors in G_i's forms a 1-factor of $G(cd^m, d)$. We have successfully constructed the hamiltonian decomposition of $G(cd^m, d)$ in this case.

Now let us prove that the hamiltonian decomposition presented satisfies the conditions C1.1 and C1.2. C_j passes through the edges $(v_{k_j+1}^0, v_{k_j+1}^1)$

and $(v^1_{k_j+1}, v^2_{k_j+1})$. Eventually in $G(cd^m, d)$, they are adjacent edges of size 1. X passes through the edges $(v^{d-3}_{k_j}, v^{d-2}_{k_j})$ and $(v^{d-2}_{k_j}, v^{d-1}_{k_j})$. X also satisfies the condition C1.1. The three vertices associated with C_j are different from the three vertices associated with X. From the fact that any pair C^i_j and $C^i_{j'}$ of hamiltonian cycles in the decomposition of G^i satisfies the condition C1.2, the three vertices $\{v^0_{k_j+1}, v^1_{k_j+1}, v^2_{k_j+1}\}$ associated with C_j are disjoint from the three vertices $\{v^0_{k_{j'}+1}, v^1_{k_{j'}+1}, v^2_{k_{j'}+1}\}$ associated with $C_{j'}$. Thus we have a hamiltonian decomposition of $G(cd^m, d)$ satisfying C1.1 and C1.2.

Case 2 odd d

The proof of this case is similar to that of Case 1. C_j and X after the edge exchange operation are shown below(See Fig. 4 (b)).

$$\begin{gathered} C_j = v^0_{k_j}, v^1_{k_j}, P^1, v^1_{k_j+2} v^2_{k_j+2}, P^2, v^2_{k_j}, \cdots, v^{d-2}_{k_j}, P^{d-2}, v^{d-2}_{k_j+2}, v^{d-1}_{k_j+2}, \\ P^{d-1}, v^{d-1}_{k_j}, v^{d-1}_{k_j+1}, v^{d-2}_{k_j+1}, \cdots, v^2_{k_j+1}, v^1_{k_j+1}, v^0_{k_j+1}, v^0_{k_j+2}, P^0 \\ X = v^0_{k_j}, v^0_{k_j+1}, v^{d-1}_{k_j}, v^{d-2}_{k_j}, v^{d-2}_{k_j+1}, v^{d-2}_{k_j+2}, \cdots, v^2_{k_j+2}, v^2_{k_j+1}, v^2_{k_j}, v^1_{k_j}, \\ v^1_{k_j+1}, v^1_{k_j+2}, v^0_{k_j+2}, v^{d-1}_{k_j+1}, v^{d-1}_{k_j+2}, P \end{gathered}$$

C_j and X are hamiltonian cycles in $G(cd^m, d)$. C_j passes through edges $(v^0_{k_j+1}, v^1_{k_j+1})$ and $(v^1_{k_j+1}, v^2_{k_j+1})$, and X passes through $(v^{d-1}_{k_j+1}, v^0_{k_j+2})$ and $(v^0_{k_j+2}, v^1_{k_j+2})$. It is easy to check that the hamiltonian decomposition presented satisfies the conditions C1.1 and C1.2, and thus omitted here.

3.2 Case of $d = 2, 3$

We show that $G(cd^m, d)$ has a hamiltonian decomposition which satisfies the following two conditions C2.1 and C2.2.

C2.1 For every hamiltonian cycle C_j in the decomposition, there exists an index k_j such that C_j passes through an edge (v_{k_j}, v_{k_j+1}) of size 1.
C2.2 For any pair C_j and $C_{j'}$ of hamiltonian cycles in the decomposition, $\{v_{k_j}, v_{k_j+1}\} \cap \{v_{k_{j'}}, v_{k_{j'}+1}\} = \emptyset$.

We observe that $G(cd^m, d)$ with degree two or three has a hamiltonian decomposition satisfying conditions C1.1 and C1.2, and that the decomposition also satisfies the above two conditions C2.1 and C2.2. We have two cases.

Case 1 $d = 3$

C^i_j and X are shown in the following. Here P^i is a path from $v^i_{k_j+1}$ to $v^i_{k_j}$ passing through all the vertices in G^i excluding the start and end vertices. P is a path from $v^2_{k_j+1}$ to $v^0_{k_j}$ passing through all the vertices $\{v^i_j \,|\, 0 \le i < d, 0 \le j < cd^{m-1}, j \ne k_j, k_j + 1\}$ in $G(cd^m, d)$.

$$\begin{gathered} C^i_j = v^i_{k_j}, v^i_{k_j+1}, P^i \\ X = v^0_{k_j}, v^1_{k_j}, v^2_{k_j}, v^0_{k_j+1}, v^1_{k_j+1}, v^2_{k_j+1}, P \end{gathered}$$

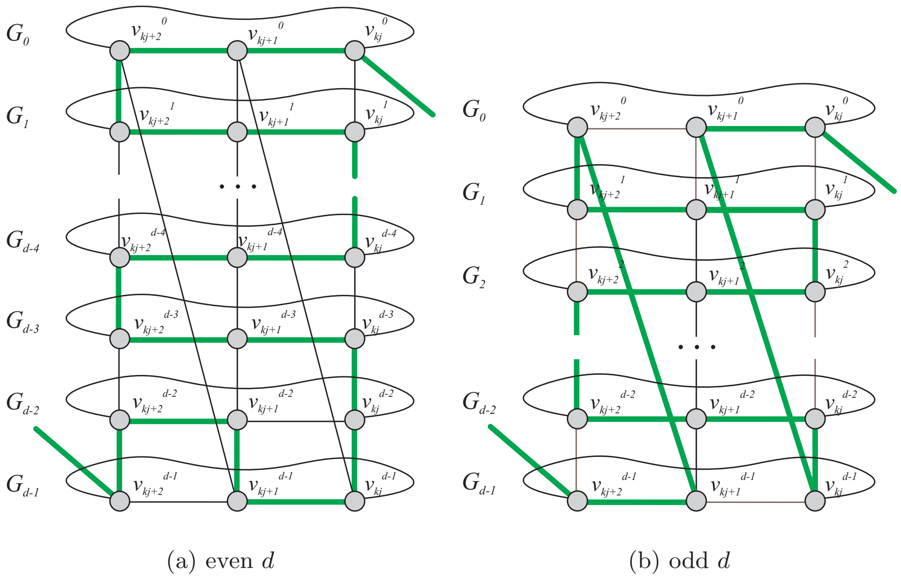


Fig. 4. Case of $d \geq 4$

C_j and X after the edge exchange operation are as follows(See Fig. 5 (a)).

$$C_j = v^0_{k_j}, v^1_{k_j}, P^1, v^1_{k_j+1}, v^2_{k_j+1}, P^2, v^2_{k_j}, v^0_{k_j+1}, P^0$$
$$X = v^0_{k_j}, v^0_{k_j+1}, v^1_{k_j+1}, v^1_{k_j}, v^2_{k_j}, v^2_{k_{j+1}}, P$$

C_j passes through edge $(v^0_{k_j}, v^1_{k_j})$ of size 1, and X passes through edge $(v^0_{k_j+1}, v^1_{k_j+1})$. The vertices $\{v^0_{k_j}, v^1_{k_j}\}$ associated with C_j are disjoint from the vertices $\{v^0_{k_{j'}+1}, v^1_{k_{j'}+1}\}$ associated with any other hamiltonian cycle $C_{j'}$. Thus the hamiltonian decomposition satisfies the conditions C2.1 and C2.2.

Case 2 $d = 2$

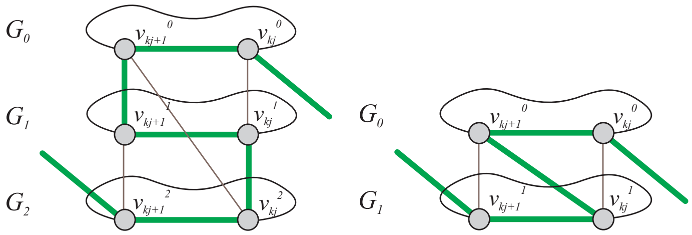
C^i_j and X are shown in the following. P^i and P can be defined in a similar way to Case 1.

$$C^i_j = v^i_{k_j}, v^i_{k_j+1}, P^i$$
$$X = v^0_{k_j}, v^1_{k_j}, v^0_{k_j+1}, v^1_{k_j+1}, P$$

C_j and X after the edge exchange operation are as follows(See Fig. 5 (b)).

$$C_j = v^0_{k_j}, v^1_{k_j}, P^1, v^1_{k_j+1}, v^0_{k_j+1}, P^0$$
$$X = v^0_{k_j}, v^0_{k_j+1}, v^1_{k_j}, v^1_{k_j+1}, P$$

C_j passes through edge $(v^0_{k_j}, v^1_{k_j})$ of size 1. But all edges in X associated with the edge exchange operation are not of size 1. If we can choose a vertex v^0_h in G_0 such that $v^0_h \neq v^0_{k_j}, v^0_{k_j+1}$ over all cycles C^0_j in the hamiltonian decomposition of G_0, then the edge (v^0_h, v^1_h) must be in X and of size 1. The existence of such a vertex can be shown by a counting argument. G_0 is a copy of $G(cd^{m-1}, d)$ with $d = 2$, that is, $G(2^{m-1}, 2)$. $G(2^{m-1}, 2)$ has degree $2(m-1) - 1 = 2m - 3$ and $\lfloor (2m-3)/2 \rfloor = m - 2$ hamiltonian cycles in the decomposition. Two vertices $v^0_{k_j}$ and $v^0_{k_j+1}$ are associated with each hamiltonian cycle C^0_j and $2(m-2)$ vertices in total. The number $2(m-2)$ is smaller than the number 2^{m-1} of vertices in $G(2^{m-1}, 2)$. Thus, we can always choose such a vertex v^0_h in G_0. Obviously, the hamiltonian decomposition satisfies the condition C2.2.



(a) $d = 3$ (b) $d = 2$

Fig. 5. Case of $d = 2, 3$

4 Concluding Remarks

We have shown that recursive circulant $G(cd^m, d)$ is hamiltonian decomposable. A hamiltonian decomposition can be constructed recursively by following the proof given in this paper. Associated with the Alspach's problem, we give conjectures on hamiltonian decomposability of recursive circulants and circulant graphs.

Conjecture 6 *(a) Recursive circulant $G(N, d)$ is hamiltonian decomposable. (b) Every connected circulant graph is hamiltonian decomposable.*

A directed version of the hamiltonian decomposition problem, called *dihamiltonian decomposition*, is decomposing a graph G into directed hamiltonian cycles, when we regard an edge (v, w) of G as two directed edges $\langle v, w\rangle$ and $\langle w, v\rangle$ of opposite direction. The problem has an application in the design of reliable algorithms for communication problems such as broadcasting and multicasting under the wormhole routing model[6]. We pose open problems on dihamiltonian decomposability of recursive circulants and m-cubes. Both recursive circulant $G(8,4)$ and 3-cube Q_3 are not dihamiltonian decomposable due to the fact that every 3-regular graph with a multiple of 4 vertices is not dihamiltonian decomposable[9].

Problem 7 *(a) Does recursive circulant $G(2^m,4)$ with $m \geq 4$ have a dihamiltonian decomposition?*
(b) Does m-cube Q_m with $m \geq 4$ have a dihamiltonian decomposition?

References

1. B. Alspach, "Unsolved problems 4.5," *Annals of Discrete Mathematics* **27**, p. 464, 1985. 297
2. J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, 5th printing, American Elsevier Publishing Co., Inc., 1976. 298
3. J. Bosák, *Decompositions of Graphs*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1990. 297
4. S. J. Curran and J. A. Gallian, "Hamiltonian cycles and paths in Cayley graphs and digraphs - a survey," *Discrete Mathematics* **156**, pp. 1-18, 1996. 297
5. C. C. Chen and N. F. Quimpo, "On strongly hamiltonian abelian group graphs," *Lecture Notes in Mathematics* **884** (Springer, Berlin), Australian Conference on Combinatorial Mathematics, pp. 23-34, 1980. 300
6. J.-H. Lee, C.-S. Shin, and K.-Y. Chwa, "Directed hamiltonian packing in d-dimensional meshes and its applications," in Proc. of *7th International Symposium on Algorithms and Computation ISAAC'96*, Osaka, Japan, pp. 295-304, 1996. 306
7. C. Micheneau, "Disjoint hamiltonian cycles in recursive circulant graphs," *Information Processing Letters* **61**, pp. 259-264, 1997. 298, 300
8. J.-H. Park and K.-Y. Chwa, "Recursive circulant: a new topology for multicomputer networks (extended abstract)," in Proc. of *IEEE International Symposium on Parallel Architectures, Algorithms and Networks ISPAN'94*, Kanazawa, Japan, pp. 73-80, Dec. 1994. 297, 298
9. J.-H. Park and H.-C. Kim, "Hamiltonian decomposition of symmetric 3-regular digraphs," in Proc. of *24th Korea Information Science Society Spring Conference*, Chunchon, Korea, pp. 711-714, Apr. 1997 (written in Korean). 306

Convertibility among Grid Filling Curves

Tetsuo Asano[1], Naoki Katoh[2], Hisao Tamaki[3], and Takeshi Tokuyama[4]

[1] School of Information Science, JAIST
Asahidai, Tatsunokuchi, 923-1292 Japan
`t-asano@jaist.ac.jp`

[2] Department of Architecture, Kyoto University
Yoshida-Honmachi, Sakyou-ku, Kyoto, 606-01 Japan
`naoki@is-mj.archi.kyoto-u.ac.jp`

[3] Department of Computer Science, Meiji University
Higashi-Mita, Tama-ku, Kawasaki, 214 Japan
`htamaki@cs.meiji.ac.jp`

[4] IBM Tokyo Research Laboratory, Yamato 242-0001, Japan
`ttoku@trl.ibm.co.jp`

1 Introduction

Consider a square grid graph G_n on n^2 grid points $\{1, \ldots, n\} \times \{1, \ldots, n\}$ in the plane, where every unit-distance pair of points are connected by an edge segment. A *grid filling curve*, or GFC for short, is a hamilton path of G_n between two corner vertices $(1, 1)$ and (n, n). Since a parity argument shows that such a hamilton path exists only if n is odd, we will assume that n is odd throughout the paper. It is easy to construct a GFC under this assumption: a standard snake-order traversal suffices. We are, however, interested in generating *random* GFCs (see below for motivations). To this end, we define a simple local operation called a *flip* that converts a GFC into another and apply a sequence of random flips to the standard GFC hoping that the result will be a random GFC in some useful sense. The first thing we would want to establish in such a random generation method is that *every* GFC has a positive probability of being generated. We show in this paper that it is indeed the case: our result states that given any two GFCs, there is a sequence of flips that converts one into the other.

Grid filling curves are not only interesting by themselves but also applied to various problems, such as efficient disk seek methods[3] and many others. One typical application among them is found in digital halftoning which is a technique to convert an image with several bits for brightness levels into a binary image consisting of black and white dots. One well-known method is the one called "error diffusion" [5] which propagates quantization errors to neighboring pixels following the order of raster scan. Unfortunately, the regularity of its scanning order produces regular patterns known as moirè in large uniform regions. To get rid of such regular patterns for better image quality we need to incorporate some randomness, which could be introduced by random grid filling curves. In fact, several methods based on this idea are proposed that use space filling curves such as Peano curve [4], Hilbert curve [7], and some others[1,2,8]. In [3] a general scheme for defining a class of space filling curves based on a grammar is presented. Experimental results suggest that those conventional discrete space filling curves or those defined by recursive grammars still retain regularities. Thus, we need really random-looking grid filling or space filling curves. Although the grammar-based approach provides some freedom in curve generation which can be exploited to introduce randomness, it suffers from the inherent constraint that (when the space is discretized into a grid) the side length of the grid must be a power of some constant. Our random generation method is free from this constraint since it is applicable to any $n \times m$ grid where either n or m is odd.

In their recent work on the hamiltonicity of solid grid graphs (which are a generalization of the grid graphs we are dealing with), Umans and Lenhart[6] adopts the following approach: start with a spanning set of cycles and apply stages of local modifications, each stage reducing the number of cycles by one, until a hamilton cycle is reached or no further reduction in the number of cycles is possible. Their basic operation is also called a flip (with a different meaning) which either merges two cycles into one or splits one cycle into two. Let us call this type of flips as Umans-Lenhart flips or UL-flips for short. If we regard a GFC as a hamilton cycle with a help of an additional edge between the two corner vertices, our flip on a GFC consists of two UL flips, one of which splits the GFC into two cycles while the other merges those two back into a GFC. Thus, we quite naturally encounter similar difficulties as encountered by those authors. Nonetheless, the two problems are quite independent, different techniques are used, and neither result implies the other. It appears that their result may imply the convertibility among GFCs via UL-flips (although the proof is not entirely clear). Even if this is the case, UL-flips are not suitable for our purposes of random generation, because a sequence of UL-flips, as opposed to a sequence of our flips, may split a GFC into a collection of an arbitrary number of cycles. In our random generation context, this means that we would do a random walk in a space in which the set of GFCs forms only a tiny subspace. Our flip combines two UL-flips so that we always stay within this subspace. Thus, convertibility via our flips is not only logically stronger than convertibility via UL-flips but also very essential in our applications.

We believe that our result can be extended to the case where the entry and exit points of a GFC are arbitrarily given as long as the parity condition is met. Such a generalization would be meaningful when we try to integrate our method into a recursive approach where a GFC of a subgrid is used as a component of a GFC of the entire grid. We have not, however, worked out technical details yet.

2 Notation and Definitions

Our grid graph G_n, as defined in the introduction, is a plane graph with $(n-1) \times (n-1)$ square faces. These faces will be referred to as *cells*. The cell with vertices (i, j), $(i+1, j)$, $(i, j+1)$, $(i+1, j+1)$ is identified with the index (i, j) so that the set of cells is $\{1, \ldots, n-1\} \times \{1, \ldots, n-1\}$. For convenience we add cells around the original plane graph, which results in an extended set of cells $C_n = \{0, 1, \ldots, n\} \times \{0, 1, \ldots, n\}$. We call a cell (i, j) a *boundary cell* if either $i \in \{0, n\}$

or $j \in \{0, n\}$; otherwise it is an *internal cell.* The *cell graph* is a graph on the vertex set C_n in which two cells (i, j) and (i', j') are adjacent if they are adjacent in the usual sense, i.e., $|i - i'| + |j - j'| = 1$.

Each GFC may be associated with a coloring of cells in the following manner. A *cell-coloring* $A : C_n \to \{black, white\}$ assigns one of the two colors to each cell, with the following fixed coloring of the boundary cells: each cell $(0, j)$ in the 0th row and each cell (i, n) in the nth column except for cell (n, n) are colored black and each cell (n, j) in the nth row and each cell $(i, 0)$ in the 0th column except for cell $(0, 0)$ are colored white. A *cross* in a cell-coloring is a set of four cells $c_1 = (i, j)$, $c_2 = (i, j + 1)$, $c_3 = (i + 1, j)$, $c_4 = (i + 1, j + 1)$ forming a square such that c_1 and c_4 are of the same color and c_2 and c_3 are of the other color (Figure 1). In a cell-coloring A, a cycle of the cell graph is called a *black* (*white*, resp.) cycle of A if A colors every cell in the cycle black (white, resp.); it is called a *monochromatic* cycle of A if it is either a black or white cycle of A. A *well-formed* cell-coloring, or WFC for short, is a cell-coloring that does not have either a cross or a monochromatic cycle. Observe that the subgraph of the cell graph induced by all the black (white) cells of a WFC is connected (because otherwise there would be a cycle of the opposite color) and hence is a tree.

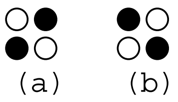


Fig. 1. Forbidden subarrays: crosses.

Proposition 1. *GFCs and WFCs are in one-to-one correspondence in the following manner.*
(1) Let P be a GFC. As we follow P from the entry point $(1, 1)$ to the exit (n, n), color the internal cells to the left of P black and those to the right of P white. Then, together with the fixed coloring of the boundary cells, the resulting cell-coloring is a WFC.
(2) Let A be a WFC. Then, the set of edges of the grid graph G_R which border two differently colored cells form a GFC.

See Figure 2. Following a convention in computer vision, the $(0, 0)$ cell is placed at the upper-left corner in our illustrations. In the sequel, we will work on WFCs rather than directly on GFCs, because WFCs visualize our definitions and facilitate our reasoning.

3 Convertibility by Successive Flips

Our local operation on GFCs is best described as a *flip* on WFCs. A *flip* is an unordered pair $\{c, d\}$ of distinct internal cells $c, d \in C_n$. We say that flip $\{c, d\}$ is

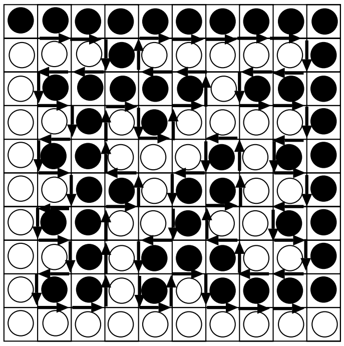


Fig. 2. A well-formed coloring and the corresponding grid filling curve.

applicable to a WFC A, if $A(c) \neq A(d)$ and swapping the values of $A(c)$ and $A(d)$ in A results in another WFC. The goal of this section is to show that, given any two WFCs A and B, there is a sequence of flips which, when applied successively to A, produces B. To this end, we define the canonical WFC, which we denote by A^*, as follows: for each internal cell (i, j), $A^*(i, j)$ is black if and only if i is even (see Figure 3).

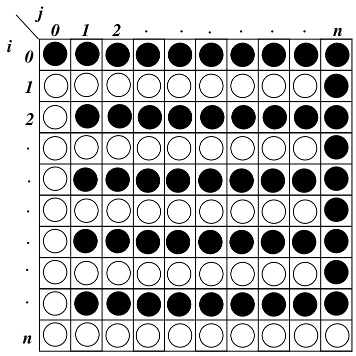


Fig. 3. Canonical WFC.

Theorem 1. *Given any WFC A, there is a sequence of at most n^3 flips that transforms A into the canonical form A^*.*

The rest of this section is devoted to the proof of the above theorem. We need some more definitions. Totally order the cells of C_n lexicographically: $(i, j) < (i', j') \Leftrightarrow i < i'$ or ($i = i'$ and $j < j'$). For a WFC A, we say that a cell c is *fixed in A* if either c is a boundary cell or an internal cell satisfying $A(c') = A^*(c')$ for every $c' \leq c$; otherwise cell c is *floating*. We call a flip $\{c, d\}$ *valid* for A

if it is applicable to A and both c and d are floating in A. Our strategy is to fix the cells one by one in the lexicographically increasing order of cells until all the cells are fixed. Thus, if c is the lexicographically smallest floating cell in the current cell-coloring A, we seek a sequence of valid flips that leads to a cell-coloring in which c is fixed. This is unfortunately not a simple task as it might seem. See Figure 4 for example. In this figure, the black cell $c = (1, 7)$ is the smallest floating cell. There is no d such that $\{c, d\}$ is a valid flip. We may construct a sequence of flips, starting from the flip of two horizontally adjacent cells in the center, that eventually inverts c. Note that *every* flip immediately applicable to this example inverts at least one cell that already has the right color in the canonical coloring (so that any single flip does not make an obvious progress towards the canonical coloring).

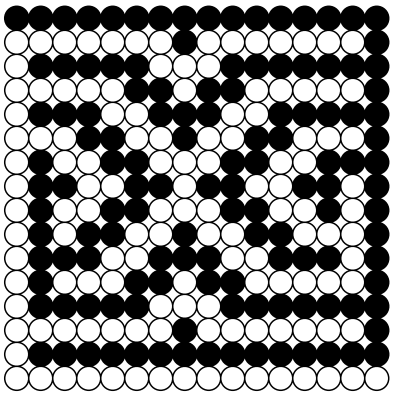


Fig. 4. An instance of WFC in which fixing the lexicographically smallest next cell requires more than one flips.

This example suggests that a cell of a particular type is of interest, which we call a *spike*. A *spike* of cell-coloring A is a cell $c = (i, j)$ such that all the three cells $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$ have the same color as c in A, unless they belong to the leftmost or the rightmost column (we allow a cell among the three to be of the opposite color if it belongs to the 0th or the nth column). Note that if $c = (i, j)$ is a spike then the two cells $(i, j-1)$, $(i, j+1)$ next to c are of the opposite color to c. See Figure 5(a). We say that a spike $c = (i, j)$ of A is *robust* if all the three cells $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$ are of the same color as c (even if they are boundary cells) and, in addition, cell $(i-1, j-2)$ or $(i-1, j+2)$ exists and has the opposite color of c. Note that $A(i-1, j-2) \neq A(c)$ ($A(i-1, j+2) \neq A(c)$, resp.) implies $A(i, j-2) \neq A(c)$ ($A(i, j+2) \neq A(c)$, resp.) since otherwise A would contain a cross. See Figure 5(b). Finally, we call a spike $c = (i, j)$ *short* if cell $(i+1, j)$ below c has the opposite color to c. Otherwise call it *long*.

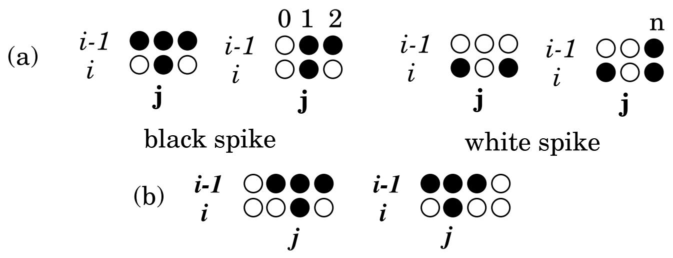


Fig. 5. (a) spike and (b) robust spikes.

Proposition 2. *Every spike of a cell-coloring A is floating in A. Moreover, the lexicographically smallest floating cell of A is a spike of A.*

Let c be a spike of A. We are interested in finding a valid flip for A of the form $\{c, d\}$. To seek such a cell d, invert the color of c alone in A and let A' denote the resulting cell-coloring. Then A' is still cross-free (since c now assumes the same color as the two cells on its sides) but contains a cycle of the color opposite to the original color of c. If c is a short spike then the cycle introduced is the 6-cycle formed by the six cells (i', j'), $i \le i' \le i+1$, $j-1 \le j' \le j+1$ (Figure 6(a)). If c is a long spike then the cycle introduced is unique and encloses at least one cell of color $A(c)$ in its inside (Figure 6(b)). In either case, we will refer to this introduced cycle as the cycle associated with spike c.

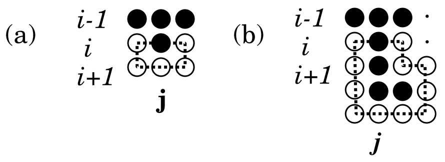


Fig. 6. (a)6-cycle associated with a short spike and (b)cycle associated with a long spike.

Let c be a spike of WFC A and let Q be its associated cycle. If d is a cell such that $\{c, d\}$ is a valid flip for A, then d must belong to Q because otherwise the monochromatic cycle Q would survive in the well-formed cell-coloring that results from applying $\{c, d\}$ to A. We call such d an *opener* of Q, because inverting d opens up the cycle Q. We want to find an opener or, when necessary, apply preliminary sequence of flips to produce an opener of Q.

Proposition 3. *Let c be a long spike of a WFC A and let Q be its associated cycle. Let $d \neq c$ be a cell on Q and let d_1 and d_2 be the two cells adjacent to d but not on Q. Then, d is an opener of Q if and only if*

(S1) d is floating,
(S2) d_1, d, and d_2 all lie on the same row or column, and
(S3) d_1 and d_2 have the color opposite to that of d.

We call a cell $d \neq c$ on Q a *potential opener* of Q if it satisfies conditions (S1) and (S2) but not necessarily (S3). We are particularly interested in potential openers for which
(1) the cells d_1, d, and d_2 in the condition (S2) above are in the same column, i.e., $d = (i, j)$, $d_1 = (i - 1, j)$ and $d_2 = (i + 1, j)$, and
(2) the color of d_2 is the same as the color of d.
Clearly, d_2 is a spike when these conditions are satisfied: we say that potential opener d has spike d_2 in this case.

For cell $c = (i, j)$, let $\mathsf{is}(c) = i + j$ denote the *index sum* of the cell.

Proposition 4. *Let c be a long spike of a WFC A and let Q be its associated cycle. Let $s = \max\{\mathsf{is}(d) \mid d$ is an internal cell belonging to $Q\}$ and suppose $\mathsf{is}(c) \leq s - 2$. Then, Q has a potential opener d with $\mathsf{is}(d) \geq s - 1$ which either is indeed an opener or has a robust spike.*

Proof. Let $c_1 = (i, j)$ denote the internal cell of Q with $\mathsf{is}(c_1) = s$ such that its row number is the largest possible under this condition. First suppose that c_1 is in the $(n - 1)$st row, i.e., $i = n - 1$. In this case, clearly c_1 and its left neighbor $(n - 1, j - 1)$ are greater than c in the lexicographic order of the cells and thus are floating. If the color of Q is white then the cycle contains the boundary cells in the bottom row and hence the cells $(n - 1, j \pm 1)$ on both sides of c_1 must be black in order to avoid a white 4-cycle (Figure 7(a)). Therefore, c_1 is indeed an opener. If the color of the cycle Q is black, then cell $(n - 2, j)$ above c_1 and cell $(n - 1, j - 1)$ on the left of c_1 belong to Q together with c_1 (because otherwise Q must contain a cell with index sum strictly greater than that of c_1 contradicting the choice of c_1) and hence are black. This implies that cell $(n - 2, j - 1)$ on the upper left of c_1 must be white in order to avoid a black 4-cycle (Figure 7(b)). Therefore the left neighbor $(n - 1, j - 1)$ of c_1 is an opener. Now suppose that $i < n - 1$. Then, cell $(i + 1, j)$ or $(i, j + 1)$ is not contained in Q because, by assumption, c_1 is the internal cell with the largest index sum in Q. Moreover, cell $(i + 1, j - 1)$ is not contained in Q because c_1 is assumed to have the largest row number among the internal cells with the largest index sum in Q. Therefore, the situation should look (assuming without loss of generality that Q is black) as shown in Figure 7(c).

If the color of cell $(i - 1, j + 1)$, which is marked by $\bullet$ in the figure, is white then cell $(i - 1, j)$ is an opener satisfying the condition. So, suppose the color of cell $(i - 1, j + 1)$ is black. Then the color of cell $(i, j + 1)$, marked by $\circ$, must be white in order to avoid a black 4-cycle. Now, cell $(i, j - 1)$ is a potential opener and if it is not indeed an opener then cell $(i + 1, j - 1)$, marked by $\diamond$, must be a spike. Because the color of cell $(i, j + 1)$ is white, it is a robust spike.

We are now ready to state and prove our main lemma for the theorem.

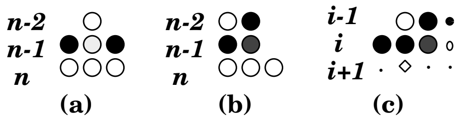


Fig. 7. Situations in the proof of Proposition 4. Cell c_1 is white and lightly shaded in (a), is black and painted dark in (b) and in (c).

Lemma 1. *Let A be a non-canonical WFC and let c be a spike of A which is either the lexicographically smallest floating cell of A or a robust spike of A. Then, there is a sequence of flips $\{c_1, d_1\}$, $\{c_2, d_2\}$, ..., $\{c_m, d_m\}$ and a sequence of WFCs $A = A_0$, A_1, A_2, ... A_m such that*

1. *A_k is the result of applying flip $\{c_k, d_k\}$ to A_{k-1}, for $1 \le k \le m$,*
2. *c_k and d_k are floating in A, for $1 \le k \le m$,*
3. *$\mathsf{is}(d_k) > \mathsf{is}(c_k)$ for $1 \le k \le m$ and $\mathsf{is}(c_k) > \mathsf{is}(d_{k+1})$ for $1 \le k < m$, and*
4. *$c_m = c$.*

Remark Note that this sequence inverts the color of c so that if c is the smallest floating cell then it is fixed as a result.

Proof. The proof is by induction on $2n - \mathsf{is}(c)$. The base case trivially holds because if $\mathsf{is}(c) \ge 2n - 1$ then c cannot be a spike. Let $c = (i, j)$ be a spike with $\mathsf{is}(c) \le 2n-2$ that is either the smallest floating cell or an arbitrary robust spike. Let Q be its associated cycle.
(Case 1: spike c is long.) We claim that, the cycle Q associated with spike c contains a cell whose index sum is at least $\mathsf{is}(c)+2$. To see this, first note that Q separates cell $(i + 1, j)$ below c from cell $(i - 1, j)$ above c. If cell $(i + 1, j)$ is enclosed inside Q, then the claim is obvious. So, suppose $(i - 1, j)$ is inside Q. This is impossible if c is the lexicographically smallest floating cell, because then inverting c will make the coloring down to row i identical to that of the canonical coloring and leaves no monochromatic cycle in that part. Therefore, c is a robust spike in this case. Therefore, cell $(i - 1, j + 1)$ is of the same color as $(i - 1, j)$ and is also enclosed inside Q; Q must contain a cell with index sum $\mathsf{is}(c) + 2$.

If Q contains an *internal cell* with index sum at least $\mathsf{is}(c) + 2$ then, by Propositions 4, Q has a potential opener d with $\mathsf{is}(d) \ge \mathsf{is}(c) + 1$ such that d is indeed an opener or has a robust spike d_1. If d is indeed and opener, a single flip $\{c, d\}$ proves the lemma, while if d_1 is a robust spike then we may apply the induction hypothesis to the pair (A, d_1) and append the flip $\{c, d\}$ to the resulting sequence to obtain the sequence for the original pair (A, c).

It remains to examine the special case where the only cell in Q with index sum at least $\mathsf{is}(c)+2$ is a boundary cell. This happens only when $j = n-2$ and the color of c is white, so the situation looks as in Figure 8(a). We let $d = (i, n - 1)$, the cell to the right of c, and a single flip $\{c, d\}$ does the task.
(Case 2: spike c is short.) In this case, the cycle Q associated with spike c is a 6-cycle formed by cells (i', j'), $i \le i' \le i + 1$, $j - 1 \le j' \le j + 1$. We proceed according to the following subcases.

(Case 2.1) $c = (i, j)$ is a robust spike. Then, we cannot have $i = n - 1$. To see this, suppose $i = n-1$. Since Q then contains boundary cells in the nth row that are white, the color of the spike c is black and, since c is a robust spike, we must have a situation as in Figure 8(b) which contains a white cycle, a contradiction. Thus, cell $d = (i+1, j)$ is an internal cell. If $d_1 = (i+2, j)$ is not a spike then flip $\{c, d\}$ is applicable to A so we are done (see Figure 8(c)). If d_1 is a spike, then we argue that it must be robust. Assume without loss of generality that cell c is black. Since c is a spike, cell $(i, j-1)$ on the left of c is white. Moreover, from the assumption that c is a robust spike, either cell $(i-1, j-2)$ or cell $(i-1, j+2)$ is white; assume that cell $(i-1, j-2)$ is white without loss of generality. This implies that cell $(i, j-2)$ is also white as noted before. Since $d_1 = (i+2, j)$ is a white spike, the internal cell $(i+1, j-1)$ must also be white. Summing up, the three cells $(i, j-2)$, $(i, j-1)$ and $(i+1, j-1)$ are all white, so that cell $(i+1, j-2)$ must be black in order to avoid a white 4-cycle, making d_1 a robust spike (Figure 8(d)). Therefore, we may apply the induction hypothesis to the pair (A, d_1) and append flip $\{c, d\}$ to the resulting sequence.
(Case 2.2) $c = (i, j)$ is the smallest floating cell of A. We first argue that $i \leq n-2$. To see this, suppose that $i = n-1$. Then, because Q includes cells in the nth row which are white, the color of c must be black. Since this is the right color for the $(n-1)$st row in the canonical coloring, this is a contradiction to the assumption that c is the smallest floating cell of A. Thus, cell $d = (i+1, j)$ is an internal cell. If cell $d_1 = (i+2, j)$ is not a spike, then flip $\{c, d\}$ is applicable to A so we are done. When d_1 is a spike, we only need to establish that it is a robust spike, because then the induction would work.

To verify that d_1 is robust, first suppose that $j \geq 3$. Assume without loss of generality that cell c is black. Then, the cells $(i, j-2)$ and $(i, j-1)$ (the two cells to the left of c) are white, because c is the smallest floating cell of A. Cell $(i+1, j-1)$ belongs to the 6-cycle Q associated with the spike c and therefore are also white. Therefore, cell $(i+1, j-2)$ must be black in order to avoid a white 4-cycle, making d_1 a robust spike. See Figure 8(e). Next suppose $j = 2$. Then cell c must be white. To see this, suppose c is black. Then cell $(i, 1)$ to the left of c is white and, since spike c is short, cell $(i+1, 2)$ below c is also white. Therefore, to avoid a cross, cell $(i+1, 1)$ must be white and a white 4-cycle is formed by cells $(i, 0)$, $(i, 1)$, $(i+1, 0)$, and $(i+1, 1)$, a contradiction. So suppose the color of c is white. Then, the color of the spike d_1 is black, and therefore it is indeed a robust spike because the boundary cell $(i+1, 0)$ is white. Finally suppose $j = 1$. This can only happen when i is odd and hence the color of c is black (because if i is even then the absence of white 4-cycles forces c to have the right color black for the canonical coloring). But then, since the three cells $(i+1, 0)$, $(i+1, 1)$ and $(i+2, 0)$ are white, d_1 must be black, so that d_1 is not indeed a spike. See Figure 8(f), where d_1 is marked by $\circ$. This completes the case analysis and hence the proof of the lemma.
Our theorem is a straightforward consequence of this lemma. Given an arbitrary WFC, we iteratively apply the lemma fixing cells one by one in the lexicographic order until we obtain the canonical WFC. The number of flips to be applied in order to fix one cell is at most n so that the total number of flips is at most $2n^3$.

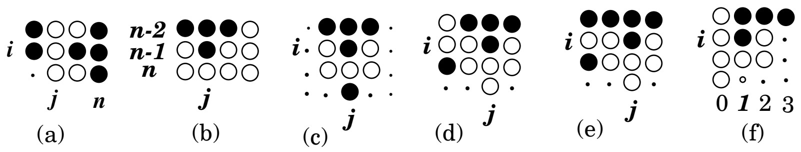


Fig. 8. Situations in the proof of Lemma 1

4 Concluding Remarks

We have shown that the space of GFCs is connected when an edge is given to each pair of GFCs that are convertible to each other by one flip. In fact, we have shown that the diameter of the graph is at most n^3. An obvious lower bound on the diameter is $\Omega(n^2)$ and it may not be very difficult to improve our upper bound into an asymptotically tight one.

Our result guarantees that, when we apply a sequence of n^3 random flips to the canonical GFC, every GFC has a positive probability of being generated. Although we have no theoretical results on the probability distribution after some number of flips are applied, an experimental study suggests that the random walk in the space of GFCs mixes quite rapidly. A theoretical analysis of the rate of mixing is an important direction of the future research.

References

1. T. Asano: "Digital halftoning algorithm based on a random space filling curve," Proc. International Conf. on Image Processing, ICIP-96, pp.545-548, Lausanne, Switzerland, 1996. 307
2. T. Asano, D. Ranjan, and T. Roos: "Digital halftoning algorithms based on optimization criteria and their experimental evaluation," *Trans. of IEICE on Fundamentals,* E-79-A, 4, pp.524-532, 1996. 307
3. T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer: "Space-filling curves and their use in the design of geometric data structure," *Theoretical Computer Science,* 181, pp.3-15, 1997. 307
4. A. J. Cole: "Halftoning without dither or edge enhancement," *The Visual Computer*, 7, pp.232-246, 1991. 307
5. R. W. Floyd and L. Steinberg: "An adaptive algorithm for spatial gray scale," *SID 75 Digest, Society for Information Display*, pp.36-37, 1975. 307
6. C. Umans and W. Lenhart: "Hamiltonian cycles in solid grid graphs," *Proc. of FOCS '97*, pp.486-505, 1997. 308
7. L. Velho and de M. Gomes: "Digital halftoning with space filling curves," *Proc. of SIGGRAPH '91*, pp.81-90, 1991. 307
8. Y. Zhang and R. E. Webber: "Space diffusion: An improved parallel halftoning technique using space-filling curves," *Proc. of SIGGRAPH '93*, pp.305-312, 1993. 307

Generalized Self-Approaching Curves*

Oswin Aichholzer[1], Franz Aurenhammer[1], Christian Icking[2], Rolf Klein[2], Elmar Langetepe[2], and Günter Rote[1]

[1] Technische Universität Graz, A-8010 Graz, Austria
[2] FernUniversität Hagen, Praktische Informatik VI, D-58084 Hagen, Germany.

Abstract. We consider all planar oriented curves that have the following property depending on a fixed angle φ. For each point B on the curve, the rest of the curve lies inside a wedge of angle φ with apex in B. This property restrains the curve's meandering, and for $\varphi \leq \frac{\pi}{2}$ this means that a point running along the curve always gets closer to all points on the remaining part. For all $\varphi < \pi$, we provide an upper bound $c(\varphi)$ for the length of such a curve, divided by the distance between its endpoints, and prove this bound to be tight. A main step is in proving that the curve's length cannot exceed the perimeter of its convex hull, divided by $1 + \cos\varphi$.

Keywords: Self-approaching curves, convex hull, detour, arc length.

1 Introduction

Let f be an oriented curve in the plane running from A to Z, and let φ be an angle in $[0, \pi)$. Suppose that, for every point B on f, the curve segment from B to Z is contained in a wedge of angle φ with apex in B. Then the curve f is called *φ-self-approaching*, generalizing the self-approaching curves introduced by Icking and Klein [5].

At the 1995 Dagstuhl Seminar on Computational Geometry, Seidel [2] posed the following open problems. Is there a constant, $c(\varphi)$, so that the length of every φ-self-approaching curve is at most $c(\varphi)$ times the distance between its endpoints? If so, how small can one prove $c(\varphi)$ to be?

Both questions are answered in this paper. We provide, for each φ in $[0, \pi)$, a constant $c(\varphi)$ with the above property, and prove it minimal by constructing a φ-self-approaching curve such that this factor $c(\varphi)$ is achieved.

Self-approaching curves are interesting for different reasons. If a *mobile robot* wants to get to the kernel of an unknown star-shaped polygon and continuously follows the angular bisector of the innermost left and right reflex vertices that are visible, the resulting path is self-approaching for $\varphi = \pi/2$; see [5]. Since the value of $c(\pi/2)$ is known to be ≈ 5.3331, as already shown in Icking et al. [6], one immediately obtains an upper bound for the competitive factor of the robot's

* This work was supported by the Deutsche Forschungsgemeinschaft, grant Kl 655/8-2, and by the Spezialforschungsbereich *Optimierung und Kontrolle*.

strategy. Improving on this, Lee and Chwa [7] give a tight upper bound of $\pi + 1$ for this factor, and Lee et al. [8] present a different strategy that achieves a factor of 3.829, while a lower bound of 1.48 is shown by López-Ortiz and Schuierer [9].

In the construction of *spanners* for euclidean graphs, one can proceed by recursively adding to the spanning structure a point from a cone of angle φ, resulting in a sequence $p_1, p_2, \ldots, p_n$ such that for each index i, $p_{i+1}, \ldots, p_n$ are contained in a cone of angle φ with apex p_i; see Ruppert and Seidel [11] or Arya et al. [3]. (Note, however, that such a sequence of points does not necessarily define a φ-self-approaching polygonal chain because the property may not hold for every point in the interior of an edge.)

Finally, such properties of curves are interesting in their own right. For example, in the book by Croft et al. [4] on open problems in geometry, *curves with increasing chords* are defined by the property that for every four consecutive points A, B, C, D on the curve, B is closer to C than A to D. The open problem of how to bound the length of such curves divided by the distance between its endpoints has been solved by Rote [10]; he showed that the tight bound equals $\frac{2}{3}\pi$. There is a nice connection to the curves studied in this paper. Namely, a curve has increasing chords if and only if it is $\pi/2$-self-approaching in both directions.

In this paper we generalize the results of [6] to arbitrary angles $\varphi < \pi$. In Sect. 2 we prove, besides some elementary properties, the following fact. Let B, C, and D denote three consecutive points on a φ-self-approaching curve f. Then

$$CD \leq BD - \cos\varphi \cdot \mathrm{length}(f[B \mathinner{..} C])$$

holds, where CD denotes the euclidean distance between points C and D and $\mathrm{length}(f[B \mathinner{..} C])$ denotes the length of f between B and C. This property accounts for the term "self-approaching"; in fact, for $\varphi \leq \pi/2$ the factor $\cos\varphi$ is not negative, so that $CD \leq BD$ holds: The curve always gets closer to each point on its remaining part. Although this property does not hold for $\varphi > \frac{\pi}{2}$, we will nevertheless see that our analysis of the tight upper bound $c(\varphi)$ directly applies to this case, too.

In Sect. 3 we show that the length of a φ-self-approaching curve cannot exceed the perimeter of its convex hull, divided by $1 + \cos\varphi$. This fact is the main tool in our analysis. It allows us to derive an upper bound for the curve's length by circumscribing it with a simple, closed convex curve whose length can be easily computed, see Sect. 4. Finally, in Sect. 5, we demonstrate that the resulting bound is tight, by constructing φ-self-approaching curves for which the upper bounds are achieved.

The proofs of Lemma 3 and Theorem 1, which are omitted or only sketched here due to the limitation of space, can be found in the full paper [1].

2 Definitions and properties

For two points B and C let $\boldsymbol{r}(B, C)$ denote the ray starting at B and passing through C. We simply write BC for the euclidean distance between B and C.

We consider plane curves $f\colon [0\,..\,1] \to \mathbb{R}^2$ parameterized by a parameter that is usually denoted by $t \in [0\,..\,1]$. It is considered as an oriented curve, i. e., it has a specified direction from beginning to end.

We do not make any assumptions about smoothness or rectifiability of the curve, although it will turn out that φ-self-approaching curves are rectifiable. For a point X on the curve we write t_X for the parameter value with $f(t_X) = X$ if no confusion arises. (We will see below in Lemma 1 that t_X is in fact unique.)

For two points $B = f(t_B)$ and $C = f(t_C)$ on f with $t_B \le t_C$, we write $f[B\,..\,C]$ or $f[t_B\,..\,t_C]$ for the part of f between B and C. By $f[B\,..]$ we denote the part of f from B to the end. By length(f), length($f[B\,..\,C]$), etc., we mean the length of the curve or of an arc.

Definition 1. *A curve f is called* φ-self-approaching *for $0 \le \varphi < \pi$ if for any point B on f there is a wedge of angle φ at point B which contains $f[B\,..]$. In other words, for any three points B, C, D with $t_B < t_C < t_D$, the angle CBD, if it is defined, is at most φ.*

Let f be an oriented curve from A to Z. Then the quantity

$$\frac{\mathrm{length}(f[A\,..\,Z])}{AZ}$$

is called the detour *of f.*

The detour of a curve is the reciprocal of the *minimum growth rate* used in [10].

We wish to bound the detour of φ-self-approaching curves. The first definition of φ-self-approaching curves also makes sense for $\varphi \ge \pi$, but then any circular arc connecting two points A and Z is φ-self-approaching, which means that the detour of such curves is not bounded. Therefore, we restrict our attention to the case $\varphi < \pi$. Each 0-self-approaching curve is a line segment and its detour equals 1.

Lemma 1. *A φ-self-approaching curve does not go through any point twice.*

Proof. Suppose the curve visits point X twice. Shortly after the first visit of X there is a point on the curve for which the φ-self-approaching property is violated. $\square$

By this lemma, we can take any point Z on the curve as the origin of a polar coordinate system and use a continuous parameterization $f(t) = (r(t), \psi(t))$ for any part $f[A\,..\,B]$ of the curve which comes before Z. For a point $X = f(t_X)$ on the curve we will use the notation $X = (r(t_X), \psi(t_X)) =: (r_X, \psi_X)$.

Lemma 2. *If $t_A < t_B < t_Z$ and $\psi_A \equiv \psi_B \pmod{2\pi}$, then $r_B \le r_A$.*

Proof. The assumption $r_B > r_A$ would lead to an angle $BAZ = \pi$, violating the φ-self-approaching property. (The case $r_B = r_A$ is also excluded, by Lemma 1, but we do not use this fact.) $\square$

The following lemma shows, roughly speaking, that a φ-self-approaching curve $f[A\,..\,Z]$ is enclosed by two oppositely winding φ-logarithmic spirals through A with pole Z, see Fig. 1. For $\varphi > \pi/2$, this is true only locally, as long as the curve does not leave the vicinity of A. In polar coordinates (r, ψ), a φ-logarithmic spiral is the set of all points with $r = e^{\psi \cot \varphi}$ or $r = e^{-\psi \cot \varphi}$.

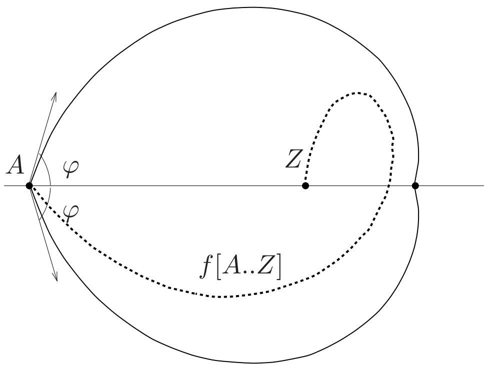


Fig. 1. For $0 < \varphi < \pi/2$ the curve is enclosed by two congruent arcs of a φ-logarithmic spiral.

Each ray through the origin intersects the spiral in the same angle φ. For $\varphi \neq \pi/2$, the length of an arc $S[A \mathrel{..} B]$ of a φ-logarithmic spiral S with pole Z is given by $\frac{1}{\cos\varphi}(AZ - BZ)$. For $\varphi = \pi/2$, the spiral degenerates to a circle. The properties of Lemma 3 are used in Sect. 4 for the circumscription of a φ-self-approaching curve by a convex area whose perimeter is easy to determine.

Lemma 3. *Let $t_A \leq t_B < t_Z$.*

(a)
$$r_B \leq r_A \cdot e^{-|\psi_B - \psi_A| \cot \varphi}.$$

(b)
$$r_B \leq r_A - \cos\varphi \cdot \operatorname{length}(f[A \mathrel{..} B]).$$

Note that Lemma 3 also applies to any subsection of the curve f.

3 φ-self-approaching curves and the perimeter of their convex hull

In this section we show that the length of a φ-self-approaching curve is bounded by the perimeter of its convex hull, divided by $1 + \cos\varphi$.

Let conv(P) denote the convex hull of a point set P and peri(P) the length of the perimeter of conv(P).

Theorem 1. *For a φ-self-approaching curve f with $0 \leq \varphi < \pi$,*

$$(1 + \cos\varphi) \operatorname{length}(f) \leq \operatorname{peri}(f).$$

The proof can only be sketched here. First, the claim is shown for a polygonal chain whose vertices lie on the curve f. This is done by induction on the number of vertices. But the length of a curve f is, by definition, the supremum of the lengths of all polygonal chains which are obtained as polygonal subdivisions of f. Therefore, an upper bound for the length of all such chains is also an upper bound for the length of f.

4 An upper bound for the detour

Theorem 2. *The length of a φ-self-approaching curve is not greater than $c(\varphi)$ times the distance of its endpoints, where*

$$c(\varphi) := \begin{cases} 1, & \text{for } \varphi = 0, \\ \max\limits_{\beta \in [0..\frac{\pi}{2}]} \dfrac{2\beta + \pi + 2}{\sqrt{5 - 4\cos\beta}}, & \text{for } \varphi = \pi/2, \\ \max\limits_{\beta \in [0..\varphi]} h(\varphi, \beta), & \text{otherwise,} \end{cases}$$

with $h(\varphi, \beta) := \dfrac{\left((1 + e^{\pi \cot\varphi}) e^{\beta \cot\varphi} - \frac{2}{1+\cos\varphi}\right) / \cos\varphi}{\sqrt{((1 + e^{\pi\cot\varphi}) e^{\beta\cot\varphi} - \cos\beta)^2 + \sin^2\beta}}$.

The function $c(\varphi)$ is strictly monotone and continuous for $\varphi \in [0, \pi)$. It tends to infinity if φ tends to π, see Fig. 2 for its values up to $\varphi = 1.8$.

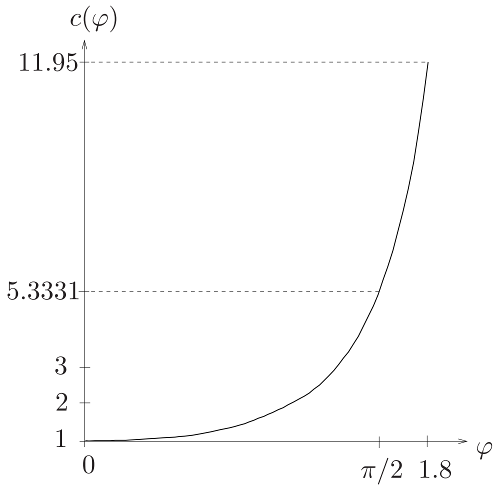


Fig. 2. The graph of $c(\varphi)$.

Proof. For $\varphi = 0$ we have a straight line segment, and the theorem is obvious. So let us assume $0 < \varphi < \pi$ from now on.

Let f be a φ-self-approaching curve from A to Z. W.l.o.g., we may assume that f does not cross the segment AZ. Otherwise we apply the bound $c(\varphi)$ successively to each subcurve between to successive curve points on AZ and add up the results. Due to the self-approaching property the curve points on AZ appear in the same order as on f, so the overall bound is less than or equal to $c(\varphi)$.

Assume w. l. o. g. that the curve starts by leaving A on the left side of the directed line AZ. Let AC be the right tangent from A to the curve, touching the curve in the point C. (The curve is on the left side of AC.) We select C as far as possible from A, and we denote the angle ACZ by β, see Fig. 3. (For $C = Z$ we set $\beta = 0$.) Note that $\beta \leq \varphi$ holds, otherwise there is a point C_0 before C on f such that the angle CC_0Z is also greater than φ which contradicts the self-approaching property at C_0.

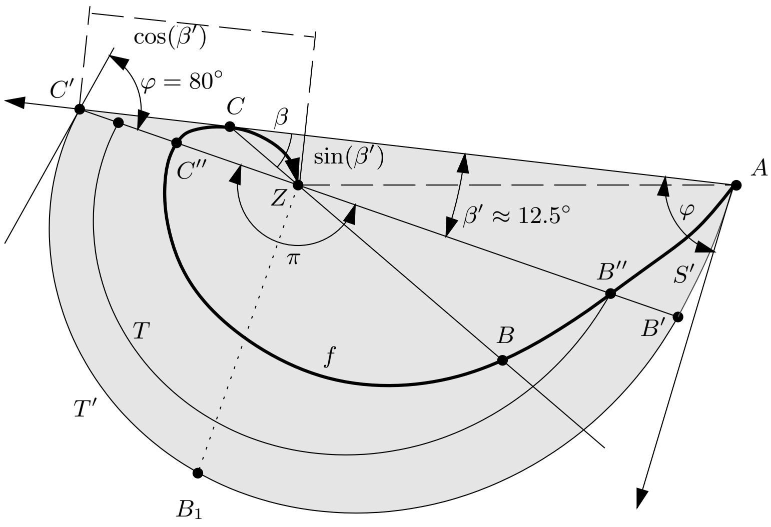


Fig. 3. A φ-self-approaching curve must lie in this area. The angle β' shown here maximizes $h(\varphi, \beta)$.

Let B denote the first point of intersection of the curve with $\boldsymbol{r}(C, Z)$. (For $C = Z$ we take $B = A$.) Since the curve neither crosses $\boldsymbol{r}(A, C)$ nor the segment AZ, Z must lie between B and C. We apply Lemma 3a to the arc $f[B \mathinner{..} C]$, considering Z as the origin, and we get

$$CZ \leq BZ \cdot e^{-\pi \cot \varphi}\,.$$

The application of the lemma is justified because, by Lemma 2, C is the first point of intersection of the curve with $\boldsymbol{r}(Z, C)$. Applying the lemma to $f[A \mathinner{..} B]$ with C as the origin, we get

$$CB \leq CA \cdot e^{-\beta \cot \varphi}\,.$$

All in all, since $CB = CZ + BZ$, we obtain

$$CA \geq CZ \cdot (1 + e^{\pi \cot \varphi}) e^{\beta \cot \varphi}\,. \tag{1}$$

Now we select a point C' on $\boldsymbol{r}(A, C)$ with $C'A \geq CA$ such that the equation

$$C'A = C'Z \cdot (1 + e^{\pi \cot \varphi}) e^{\angle AC'Z \cdot \cot \varphi} \tag{2}$$

is fulfilled. Such a point C' exists, since, as we move C' further away from A, the ratio $C'A : C'Z$ converges to 1, the angle $\beta' := \angle AC'Z$ decreases towards 0 and hence $e^{\beta' \cot \varphi}$ also converges to 1, whereas $1 + e^{\pi \cot \varphi}$ is a constant bigger than 1. Therefore the inequality (1) changes direction as $AC' \to \infty$. We have $\beta' \leq \beta \leq \varphi$, and also $C' \neq Z$.

In the following let B' be the point on $\boldsymbol{r}(C', Z)$ with $B'Z = C'Z \cdot e^{\pi \cot \varphi}$ and $\angle C'ZB' = \pi$, see Fig. 3.

Lemma 4. *A φ-self-approaching curve from A to Z is contained in the convex region bounded by the following three curves, see Fig. 3:*

(1) *a φ-logarithmic spiral from A to B' of polar angle β' with pole C';*
(2) *a φ-logarithmic spiral from B' to C' of polar angle π with pole Z;*
(3) *the segment AC'.*

Proof. Let B'' be the first intersection point of f with $\boldsymbol{r}(C', Z)$. By Lemma 3a applied to the pole C, the arc $f[A \mathinner{..} B'']$ lies inside the φ-logarithmic spiral S with pole C with polar angle β starting at A. The φ-logarithmic spiral S' with pole C' through A is obtained from S by stretching it about the center A, and hence $f[A \mathinner{..} B'']$ is also contained inside S', and in part (1) of the region boundary. In particular, $B''Z \leq B'Z$. It follows with the help of Lemma 2 that, between the rays $\boldsymbol{r}(Z, A)$ and $\boldsymbol{r}(Z, B')$, no point of the whole curve f can lie outside S'.

Now let C'' be the first intersection point of f with $\boldsymbol{r}(Z, C')$. By Lemma 3a applied to the pole Z, the arc $f[B'' \mathinner{..} C'']$ lies inside the φ-logarithmic spiral T around pole Z with polar angle π starting at B''. Since $B''Z \leq B'Z$, the arc lies inside the logarithmic spiral T' which forms part (2) of the region boundary. Again, it follows with the help of Lemma 2 that, below the line $C'B'$, no point of the whole curve f can lie outside T'.

Finally, in the region between the rays ZC' and ZA, the curve cannot escape across the segment AC', and the proof is complete. □

Now we can prove Theorem 2. We treat only the case $\varphi \neq \pi/2$, where we have proper spirals. The case $\varphi = \pi/2$, where we have circular arcs, has already been treated in [6].

We choose a scale such that $C'Z$ equals 1. Now $AC' = (1+e^{\pi \cot \varphi})e^{\beta' \cot \varphi}$ and $B'Z = e^{\pi \cot \varphi}$ holds by construction. Therefore the lengths of the three curves in Lemma 4, using the formula for the arc length of a φ-logarithmic spiral, are given by

$$\begin{aligned}
L_1 &:= \frac{1}{\cos \varphi}(AC' - B'C') = \frac{1}{\cos \varphi}\Big((1 + e^{\pi \cot \varphi})e^{\beta' \cot \varphi} - (1 + e^{\pi \cot \varphi})\Big) \\
L_2 &:= \frac{1}{\cos \varphi}(B'Z - C'Z) = \frac{1}{\cos \varphi}(e^{\pi \cot \varphi} - 1) \\
L_3 &:= AC' = (1 + e^{\pi \cot \varphi})e^{\beta' \cot \varphi}\,,
\end{aligned}$$

and for the distance of the endpoints of f we have

$$AZ = \sqrt{((1 + e^{\pi \cot \varphi})e^{\beta' \cot \varphi} - \cos \beta')^2 + \sin^2 \beta'} \, .$$

Altogether we conclude from Theorem 1

$$\frac{\text{length}(f)}{AZ} \leq \frac{L_1 + L_2 + L_3}{AZ} \cdot \frac{1}{1 + \cos \varphi} \, .$$

The right term can be transformed to

$$\frac{(1 + e^{\pi \cot \varphi})e^{\beta' \cot \varphi} - \frac{2}{1+\cos \varphi}}{\cos \varphi \sqrt{((1 + e^{\pi \cot \varphi})e^{\beta' \cot \varphi} - \cos \beta')^2 + \sin^2 \beta'}} \, . \qquad \square$$

5 Tightness of the upper bound

Theorem 3. *The upper bound $c(\varphi)$ given in Theorem 2 for the detour of φ-self-approaching curves is tight.*

Proof. We construct a φ-self-approaching curve f from A to Z as in the proof of Lemma 4, the construction is shown in Fig. 4. The curve starts with two

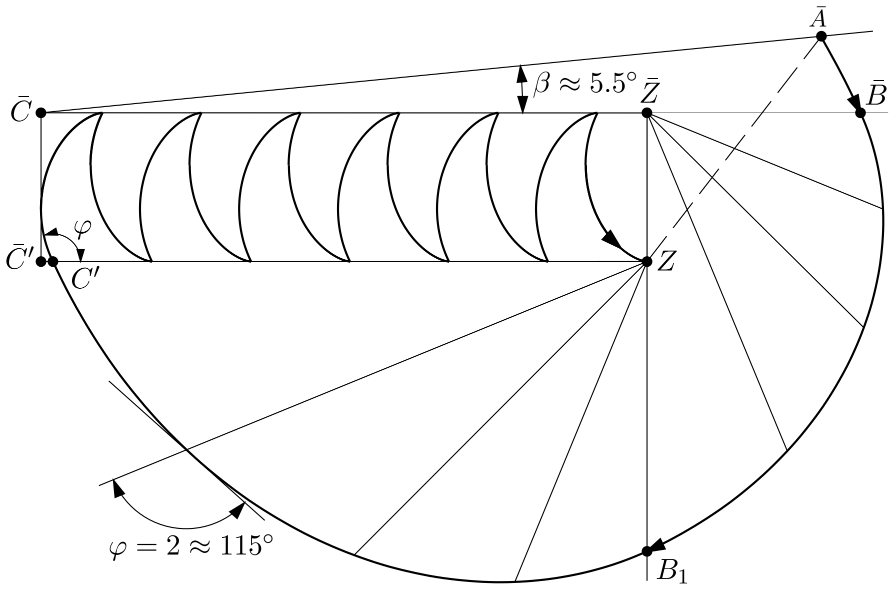


Fig. 4. The construction of the φ-self-approaching curve with maximal detour for φ=2.

logarithmic spirals similar to part (1) and part (2) in the proof of Lemma 4, except that part (2) is split into two parts. The segment $C'Z$ of length 1 is replaced by a φ-self-approaching zigzag curve of length $L_3' = \frac{2}{1+\cos\varphi}$ from C'

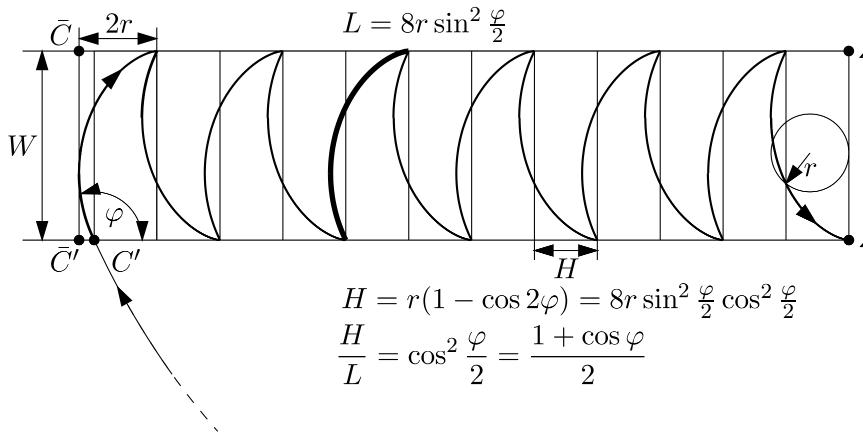

$\bar{C}$
$2r$
$L = 8r\sin^2\frac{\varphi}{2}$
$\bar{Z}$
W
r
φ
$\bar{C}'$
C'
Z
H
$H = r(1-\cos 2\varphi) = 8r\sin^2\frac{\varphi}{2}\cos^2\frac{\varphi}{2}$
$\frac{H}{L} = \cos^2\frac{\varphi}{2} = \frac{1+\cos\varphi}{2}$

Fig. 5. A sequence of cycloid parts forming a φ-self-approaching curve inside a thin rectangle.

to Z, which moves inside a thin rectangle along the segment $C'Z$. This last part of the curve between Z and C', see Fig. 5, is obtained by "stacking" small cycloids $(x,y) = (r(\alpha - \sin\alpha), r(1-\cos\alpha))$, for $0 \le \alpha \le 2\varphi$. One piece of cycloid has "height" $H = r(1-\cos 2\varphi) = 8r\sin^2\varphi\cos^2\varphi$ and length $L = 8r\sin^2\frac{\varphi}{2}$. We must choose the size parameter r in such a way that $1/H$ is an even integer n; then the curve will precisely connect the points C' and Z. The length of the curve is then $L/H = \frac{2}{1+\cos\varphi}$, which is what we need. The width of the construction is $W = r(2\varphi - \sin 2\varphi)$. We arrange the cycloid pieces on the left side of the segment $C'Z$, as indicated in Fig. 5; then the curve is contained in a rectangle $\bar{C}'Z\bar{Z}\bar{C}$ of width $W = Z\bar{Z}$. The long side $Z\bar{C}'$ slightly extends the segment ZC' by the amount $2r - H$.

Now we can describe the whole construction of the curve, in reverse direction. The curve consists of the following parts (numbered in accordance with Lemma 4, using the notation from there):

(3′) The curve ends with the φ-self-approaching curve from C' to Z described above, whose length is $\frac{2}{1+\cos\varphi}$;

(2) before, there is a φ-logarithmic spiral of polar angle $\pi/2$ with pole Z connecting C' to some point B_1.

(2′) then, a φ-logarithmic spiral of polar angle $\pi/2$ with pole $\bar{Z}$ connecting B_1 to $\bar{B}$;

(1′) the curve starts with a φ-logarithmic spiral of polar angle β with pole $\bar{C}$ between a point $\bar{A}$ and $\bar{B}$, where $\beta \le \varphi$ is the value for which the maximum in the definition of $c(\varphi)$ is attained.

It can be checked that the parts which are logarithmic spirals are always φ-self-approaching, as the line from any point X to the current pole contains the whole curve on one side. So obviously the whole curve is φ-self-approaching. Since r can be made as small as we like, we have $\bar{C} \to C'$, $\bar{Z} \to Z$, $\bar{A} \to A$, $\bar{B} \to B'$ as $r \to 0$, and the logarithmic spirals that we use will approach the "ideal" logarithmic spirals that appear in Lemma 4, see Fig. 3. This means that

$$\frac{\text{length}(f)}{\bar{A}Z} \to \frac{L_1 + L_2 + L_3'}{AZ} = \frac{(1 + e^{\pi \cot \varphi})e^{\beta \cot \varphi} - 2 + \frac{2}{1+\cos\varphi} \cdot \cos\varphi}{\cos\varphi \sqrt{((1 + e^{\pi \cot \varphi})e^{\beta \cot \varphi} - \cos\beta)^2 + \sin^2\beta}}$$

which equals $c(\varphi)$. □

6 Conclusions

Here we analyze the maximum length of curves with an upper bound on the angular wedge at A. This condition is not symmetric since it distinguishes the source and the target of the curve. One might also consider a symmetric situation, where curves are φ-self-approaching in both directions.

Generalizations to three dimensions are also completely open.

References

1. O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe, and G. Rote. φ-self-approaching curves. Technical Report 226, Department of Computer Science, FernUniversität Hagen, Germany, 1997. Submitted for publication. 318
2. H. Alt, B. Chazelle, and R. Seidel, editors. *Computational Geometry.* Dagstuhl-Seminar-Report 109. Internat. Begegnungs- und Forschungszentrum für Informatik, Schloss Dagstuhl, Germany, March 1995. 317
3. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 489–498, 1995. 318
4. H. P. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry.* Springer-Verlag, 1990. 318
5. C. Icking and R. Klein. Searching for the kernel of a polygon: A competitive strategy. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 258–266, 1995. 317
6. C. Icking, R. Klein, and E. Langetepe. Self-approaching curves. Technical Report 217, Department of Computer Science, FernUniversität Hagen, Germany, 1997. To appear in Mathematical Proceedings of the Cambridge Philosophical Society. 317, 318, 323
7. J.-H. Lee and K.-Y. Chwa. Tight analysis of a self-approaching strategy for on-line kernel-search problem. Technical report, Department of Computer Science, KAIST, Taejon, Korea, 1998. 318
8. J.-H. Lee, C.-S. Shin, J.-H. Kim, S. Y. Shin, and K.-Y. Chwa. New competitive strategies for searching in unknown star-shaped polygons. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 427–429, 1997. 318

9. A. López-Ortiz and S. Schuierer. Position-independent near optimal searching and on-line recognition in star polygons. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 445–447, 1997. 318
10. G. Rote. Curves with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 115(1):1–12, 1994. 318, 319
11. J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 207–210, 1991. 318

The Steiner Tree Problem in λ_4-geometry Plane

Guo-Hui Lin[1,2⋆] and Guoliang Xue[1⋆⋆]

[1] Department of Computer Science, University of Vermont
Burlington, VT 05405, USA
[2] Institute of Applied Mathematics, Chinese Academy of Sciences
Beijing 100080, The People's Republic of China
{ghlin,xue}@cs.uvm.edu

Abstract. In this paper, we study the Steiner tree problem in the λ_4-geometry plane in which any line, half line or line segment must go in an orientation of $\frac{i\pi}{4}$ with the positive x-axis, $0 \le i \le 7$, and the distance between two points is the length of the shortest polygonal path connecting them. We show that for any set of n terminal points, there exists a Steiner minimal tree interconnecting these terminal points such that all Steiner points are in $\mathcal{G}_{\lceil \frac{2n}{3} \rceil - 1}$, the $(\lceil \frac{2n}{3} \rceil - 1)^{st}$-generation grid points formed by the n terminal points. Our result improves previous known result which guarantees that for any set of n terminal points, there is a Steiner minimal tree in which all Steiner points are in $\mathcal{G}_{n-2}$.

1 Introduction

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n given points (called the *terminal points*) in the plane with a distance function d. A minimum cost network interconnecting these terminal points is called a *Steiner minimal tree* (SMT). The cost of a network is the sum of its edge costs and the cost of an edge is the distance (measured by the distance function d) between its end points. Therefore an SMT is always a tree network whose leaf vertices are some of the terminal points and whose internal vertices are either terminal points or some *Steiner points* which are introduced to reduce network cost. The Steiner tree problem asks for an SMT for a given set of terminal points with a given distance function. The Steiner tree problem with Euclidean and rectilinear distances have attracted much attention due to their applications in telecommunications and in VLSI design of printed circuits. For details, see [1,3,4,7].

Traditionally, wires are allowed to go horizontally and vertically in a printed circuit. Due to recent advances in VLSI technology, there have been increased interests in hexagonal routing and 45^o routing. In 1987, Widmayer, Wu and

⋆ The research of this author was supported in part by the Army Research Office grant DAAH04-96-10233, and by the Presidential Chuang Xin Foundation of the Chinese Academy of Sciences.

⋆⋆ The research of this author was supported in part by the Army Research Office grant DAAH04-96-10233 and by the National Science Foundation grants ASC-9409285 and OSR-9350540.

Wong studied some distance problems in fixed orientations [13]. Since then, many results in this area have appeared in the literature [2,8,9,10,11,12,13,14].

The λ_σ-geometry plane is a generalization of the rectilinear plane. The angle formed by a line, half line, or line segment with the positive x-axis must equal to $\frac{i\pi}{\sigma}$ for some integer i ($0 \leq i < 2\sigma$) in the λ_σ-geometry plane. Each of these 2σ orientations is called a *legal orientation*. Orientations $\frac{i\pi}{\sigma}$ and $\frac{(i+1)\pi}{\sigma}$ are said to be adjacent orientations for $i = 0, 1, \ldots, 2\sigma - 1$ (2σ is treated as 0). A *legal* polygonal line in the λ_σ-geometry is a polygonal line with each segment in a legal orientation. The distance between two points in the λ_σ-geometry plane is the Euclidean length of the shortest legal polygonal line connecting these two points.

For any two points p and q, we use $e[p, q]$ to denote a legal polygonal line connecting p to q with minimum length. If the straight line segment $\overline{pq}$ connecting p and q is in a legal orientation, then the (Euclidean) length of line segment connecting p and q is the λ_σ distance between p and q. In this case, $e[p, q]$ is unique and we say $e[p, q]$ is *straight*. Otherwise, we say $e[p, q]$ is *non-straight*. We will also say p and q are straight (non-straight) when $e[p, q]$ is straight (non-straight). If $e[p, q]$ is non-straight, the legal polygonal line connecting p to q can be realized by a *canonical path* consisting of exactly two straight line segments with orientations equal to two adjacent λ_σ-orientations. The junction point where the two segments of a non-straight edge meet is called the *turning point* of that edge. Figure 1 illustrates the two canonical paths connecting p and q (p-s-q with turning point s and p-t-q with turning point t) and a non-canonical path connecting p and q (p-u-v-w-x-q). In the rest of this paper, we will always use canonical paths as the edges connecting two points. The segment incident to p is called the *first segment* of $e[p, q]$; while the other called the *second segment* of $e[p, q]$. Note that by our definition, the edge $e[p, q]$ has a direction. The first segment of $e[p, q]$ is incident to p while the first segment of $e[q, p]$ is incident to q. When $e[p, q]$ is straight, its second segment is empty. To simplify notation, we will use the *turning point* of $e[p, q]$ to mean point q when edge $e[p, q]$ is straight. There are two ways to connect two non-straight points using a canonical path, and these two non-straight edges form a parallelogram (Figure 1). We use $para(p, q)$ to denote the parallelogram formed by the two non-straight edges connecting p and q when p and q are not straight, and the straight line segment connecting p and q otherwise. In both cases, $para(p, q)$ is the set of all points that is on some shortest legal path from p to q. One non-straight edge can be obtained from the other by a *flipping* operation [6]. In Figure 1, the edge p-s-q can be obtained by flipping the edge p-t-q and vice versa.

Given a set P of points in the λ_2-geometry (i.e., rectilinear) plane, we can draw two legal straight lines passing through each of the given points. The set of *grid points* (intersections of two lines) of these lines is called the Hanan-grid. Hanan [5] proved that there exists a rectilinear SMT whose Steiner points are all in the Hanan-grid. In a recent paper [8], Lee and Shen proved the so-called *Multi-Level Grid Theorem* which says that in λ_σ-geometry plane, $\sigma \geq 3$, i.e.,

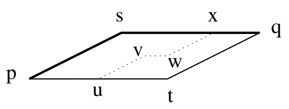


Fig. 1. The non-straight edges connecting p to q in λ_4-geometry plane.

there exists an SMT whose Steiner points are all grid points in $\mathcal{G}_{n-2}$, for $n \geq 3$, where $\mathcal{G}_k$ is defined as follows.

Let $\mathcal{G}_0$ be the set of terminal points, i.e. $\mathcal{G}_0 = P$, which we call the 0^{th} generation grid points. In general, for each grid point in $\mathcal{G}_k$, draw σ straight lines of angles $0, \frac{\pi}{\sigma}, \frac{2\pi}{\sigma}, \ldots, \frac{(\sigma-1)\pi}{\sigma}$ with the positive x-axis. The grid points produced by these $\sigma|\mathcal{G}_k|$ lines form the $(k+1)^{st}$ *generation* grid $\mathcal{G}_{k+1}$. We also use $\mathcal{G}_k(Q)$ to denote the k^{th} generation grid formed by point set Q. Using this notation, we have $\mathcal{G}_2 = \mathcal{G}_1(\mathcal{G}_1(P))$, $\mathcal{G}_3 = \mathcal{G}_1(\mathcal{G}_2(P))$, etc.

We will use T_n to denote a Steiner minimal tree with n terminal points in λ_σ-geometry, use $cost(e[p,q])$ to denote the length of edge $e[p,q]$, and use $cost(T_n)$ to denote the cost of T_n which is the sum of its edge costs. If n is known from the context, T_n will be simplified by T.

For the case where $\sigma = 3$, Du and Hwang [2], Lee *et al.* [9], and Lee and Shen [8] proved that there exists a T_n whose Steiner points are all in $\mathcal{G}_{n-2}$, for $n \geq 3$. More recently, Yan *et al.* [14] proved that there exists a T_n whose Steiner points are all in $\mathcal{G}_{\lceil \frac{n-2}{2} \rceil}$. This significantly reduces the number of Steiner point candidates. They [14] also conjectured that the bound $\lceil \frac{n-2}{2} \rceil$ holds true in λ_σ-geometry for all $\sigma \geq 4$. From now on, we will assume $\sigma = 4$ unless otherwise specified.

In this paper, we prove that for any given set of n terminal points in λ_4-geometry plane, there exists an SMT T_n whose Steiner points are all in $\mathcal{G}_{\lceil \frac{2n}{3} \rceil - 1}$. We also prove some nice structures of Steiner minimal trees in λ_4-geometry plane. These results may be useful in the design of efficient approximation algorithms. The rest of this paper is organized as follows. In Section 2, we present some basic definitions and conventions. Several preliminary facts and properties of an SMT are also presented there. In Section 3, we prove some fundamental facts about Steiner minimal trees interconnecting four terminal points. In Section 4, we prove our main result, i.e., for any given set of n terminal points, there exists a T_n whose Steiner points are all in $\mathcal{G}_{\lceil \frac{2n}{3} \rceil - 1}$. We conclude our paper in Section 5.

2 Preliminaries

Definition 1. *A* full *Steiner minimal tree is a Steiner minimal tree in which all terminal points are leaves of the tree.*

A non-full SMT can be decomposed into several full Steiner minimal subtrees whose union is the given SMT. Since we are trying to prove an upper bound on k

such that there is an SMT whose Steiner points are all in $\mathcal{G}_k$, we may assume that there is no non-full SMT for our given terminal set P, without loss of generality.

Definition 2. *In an SMT, an edge incident with two Steiner points is called a* Steiner edge*; an edge incident with one terminal point and one Steiner point is called a* terminal edge*.*

Lemma 1. *In any SMT in the λ_4-geometry plane, every Steiner point q has a degree $deg(q)$ such that $3 \leq deg(q) \leq 4$.* □

Lemma 2. *In any SMT in the λ_4-geometry plane, the number of Steiner points is at most $n-2$.* □

Lemma 3. [8,9] *There exists an SMT for a set of n terminal points such that the Steiner points are grid points in $\mathcal{G}_{n-2}$, for $n \geq 3$.* □

For ease of presentation, we denote the line going through point p in an angle of 0 with the positive x-axis as $l_0(p)$; denote the line going through point p in an angle of $\frac{\pi}{4}$ with the positive x-axis as $l_{+1}(p)$; denote the line going through point p in an angle of $\frac{\pi}{2}$ with the positive x-axis as $l_\infty(p)$; denote the line going through point p in an angle of $\frac{3\pi}{4}$ with the positive x-axis as $l_{-1}(p)$. The following two lemmas on 3-point case is of great importance.

Lemma 4. *In an SMT for a set of three points p_1, p_2 and p_3, the three first segments of the edges $e[q,p_1], e[q,p_2], e[q,p_3]$ at the Steiner point q form three angles of $\frac{\pi}{2}$, $\frac{3\pi}{4}$, $\frac{3\pi}{4}$.* □

Lemma 5. [8] *In an SMT for a set of three terminal points $\{p_1, p_2, p_3\}$, at most one of the edges incident at the Steiner point q may be non-straight. Furthermore, the second segment of the non-straight edge, say $e[q,p_2]$ (the portion of the non-straight edge that does not touch q), is not parallel to either $e[q,p_1]$ or $e[q,p_3]$.*
□

Lemma 6. *If there is only one Steiner point in the SMT for a set P of four terminal points, then the four edges incident at this Steiner point are all straight and they form a cross with the Steiner point q at the cross point,* □

Definition 3. *Let T_n be a full SMT for a set P of n terminal points. After deleting all the terminal edges together with the terminal points, we obtain a subtree of T_n interconnecting the Steiner points of T_n. A leaf in this subtree is called a* leaf Steiner point *of T_n while all other Steiner points are called* internal Steiner points *of T_n. From Lemmas 4, 5 and 6, a leaf Steiner point must be incident with two terminal edges which are not in a legal straight line. The two corresponding terminal points are said to form a* terminal pair*.*

3 The 4-point Case

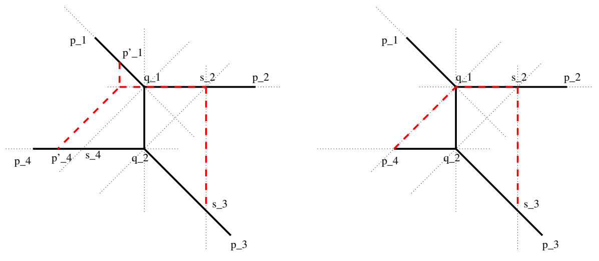
In this section, we will study the possible configurations of a full SMT interconnecting a set of four terminal points $P = \{p_1, p_2, p_3, p_4\}$. Without loss of generality, we always assume that p_1, p_2 are adjacent with Steiner point q_1 and p_3, p_4 are adjacent with Steiner point q_2, in any full SMT under consideration. In the case that q_1 coincides with q_2, that is, there is only one Steiner point in the SMT, we simply denote this Steiner point by q.

Lemma 7. *Suppose that in an SMT with two Steiner points q_1, q_2, the five edges are all straight and they form a configuration as shown in Figure 2. Then at least one of the following three inequalities is true:*

1. $|q_1p_2| < |q_1q_2|$;
2. $|q_2p_4| \le |q_1q_2|$;
3. $|q_2p_3| < \sqrt{2}|q_1q_2|$.

Furthermore, if $|q_2p_4| = |q_1q_2|$ and $|q_2p_3| \ge \sqrt{2}|q_1q_2|$, then $|q_1p_2| \le |q_1q_2|$.

Proof. Suppose to the contrary that none of the three inequalities holds. Let s_2 be the intersection point of lines $l_0(q_1)$ and $l_{+1}(q_2)$. Let s_4 be the intersection point of lines $l_0(q_2)$ and $l_{+1}(q_1)$. It follows that these two points are on edge $e[q_1, p_2]$ and edge $e[q_2, p_4]$, respectively. Furthermore, let s_3 be the intersection point of lines $l_{-1}(q_2)$ and $l_\infty(s_2)$, then s_3 lies on edge $e[q_2, p_3]$.



(a) (b)

Fig. 2. Proof of Lemma 7.

Now, let p'_4 be a point in the segment $\overline{p_4s_4}$ such that $\overline{p'_4s_4}$ has a length small enough to enable p'_1 to be on the edge $e[q_1, p_1]$ ($|q_1p'_1| = \sqrt{2}|p'_4s_4|$), see Figure 2(a). Consider the induced subtree in the initial SMT which interconnects the points p'_1, s_2, s_3 and p'_4 and the tree consisting of dashed line segments interconnecting these four points. It is obvious that the latter has a length strictly less than that of the former, since $\overline{p'_4s_4}$ has a positive length. This contradicts the definition of an SMT. Therefore, at least one of the three inequalities should hold.

Suppose that $|q_2p_4| = |q_1q_2|$ and $|q_2p_3| \geq \sqrt{2}|q_1q_2|$. If $|q_1p_2| > |q_1q_2|$, we may use line segments $\overline{p_4q_1}$, $\overline{q_1s_2}$ and $\overline{s_2s_3}$ to replace the subtree induced by points p_4, q_2, q_1, s_2, s_3 in the initial SMT, see Figure 2(b). This new tree is also an SMT. However, since the three angles around s_2 are not $\frac{\pi}{2}$, $\frac{3\pi}{4}$, and $\frac{3\pi}{4}$, this is in contradiction with Lemma 4. Therefore, $|q_1p_2| \leq |q_1q_2|$. □

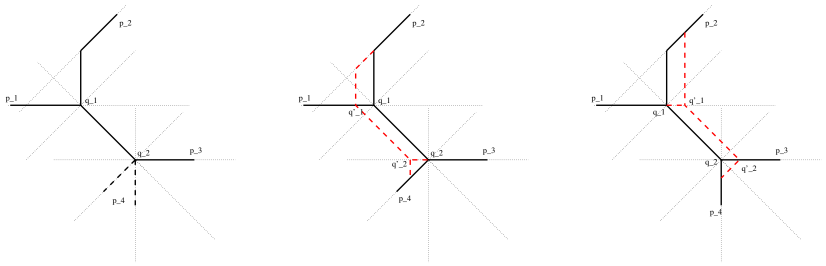
In many of our proofs in this paper, we will transform an SMT T_n interconnecting a set P of terminal points into another SMT T_n' interconnecting the set P of terminal points. Usually the transformation is done by moving some of the Steiner points while keeping the connections, i.e., if there is an edge connecting points p and q in T_n, there is also an edge connecting p' and q' in T_n', where p' is the new position of p in T_n' and q' is the new position of q' in T_n' (note that a terminal point never changes position). By *transforming ε amount from edge e_1 to edge e_2*, we mean a transformation which decreases the length of e_1 by ε and increases the length of e_2 by ε while all other edges keep their shapes (either straight or non-straight). Note that we allow ε to be positive as well as negative.

Lemma 8. *In an SMT with two Steiner points, suppose edge $e[q_1, p_2]$ is non-straight. Then we can always transform $e[q_1, p_2]$ by length ε to a terminal edge incident at q_2, (that is $e[q_2, p_4]$ in Figure 3) without changing the total length of the tree.*

Proof. Since $e[q_1, p_2]$ is non-straight, from Lemma 5, we derive that there is only one possible configuration (under reflection and rotation) for the four points q_1, p_1, p_2, q_2. Without loss of generality, assume their relative positions are as shown in Figure 3(a). Notice that although the relative positions of these four points may vary, the angles formed by the three edges are fixed by Lemma 4. It follows that $\angle_{p_1q_1q_2} = \frac{3\pi}{4}$.

For ease of presentation, an edge $e[s, t]$ with its first segment in direction α (with the positive x-axis) will be said to go out of point s in direction α. For example, in Figure 3(a), $e[q_1, p_1]$ goes out of q_1 in direction π, $e[q_1, p_2]$ goes out of q_1 in direction $\frac{\pi}{2}$, while $e[q_1, q_2]$ goes out of q_1 in direction $\frac{7\pi}{4}$. Using this terminology, we can say that there is no edge going out of q_2 in directions $\frac{\pi}{4}, \frac{\pi}{2}, \pi$ and $\frac{7\pi}{4}$ (by Lemma 4). It follows that there should be a straight edge going out of q_2 in direction 0, denoted by $e[q_2, p_3]$. In the figure, the dashed line segments incident at q_2 stand for possible edges (but only one, denoted by $e[q_2, p_4]$) going out of q_2.

Suppose now there is an edge going out of q_2 in direction $\frac{5\pi}{4}$. We can transform $e[q_1, p_2]$ by length of ε ($\varepsilon < 0$ in this case) to edge $e[q_2, p_4]$ by sliding line segment $\overline{q_1q_2}$ along $e[q_1, p_1]$ in direction π and then updating the tree using the dashed line segments (see Figure 3(b)). Suppose there is an edge going out of q_2 in direction $\frac{3\pi}{2}$. We can transform $e[q_1, p_2]$ by length of ε ($\varepsilon > 0$ in this case) to edge $e[q_2, p_4]$ by sliding line segment $\overline{q_1q_2}$ along $e[q_2, p_3]$ in direction 0 and then updating the tree (see Figure 3(c)), similarly, using the dashed line segments. □



(a) the possible configuration

(b) $\varepsilon = (\sqrt{2} - 2)|q_1q_1'| < 0$

(c) $\varepsilon = (2 - \sqrt{2})|q_1q_1'| > 0$

Fig. 3. Proof of Lemma 8.

Remark 1. By increasing $|\varepsilon|$, the transforming process done in the proof of Lemma 8 will result in one of the following three possible configurations:

1. $e[q_1, p_2]$ becomes straight; or else, one of the following two:
2. If $\overline{q_1q_2}$ is slid in direction π, then $e[q_2, p_4]$ becomes straight and goes in direction $\frac{3\pi}{2}$ or q_1 is moved to coincide with p_1;
3. If $\overline{q_1q_2}$ is slid in direction 0, then $e[q_2, p_4]$ becomes straight and goes in direction $\frac{5\pi}{4}$ or q_2 is moved to coincide with p_3.

Corollary 1. *If there are two non-straight edges in an SMT for a set of four terminal points, we can straighten one of them without increasing the tree length.*

4 The General Case

Lemma 9. [8] *Suppose the two closest terminal pairs in a full SMT for a set of n terminal points are connected by a Steiner path $q_1, q_2, \cdots, q_{k+1}$, $k \geq 1$. Assume that each terminal pair has a non-straight terminal edge, and the other terminal edges are all straight. We can transform a non-straight edge by length ε from one end of the Steiner path to the other without changing the total length of the SMT, provided that k is even (odd) and the two non-straight edges are on the same side (different sides) of the Steiner path.*

This lemma was stated in [8] without detailed proof. Since it plays an important role in our later arguments, we will provide a proof of it. In fact, we will prove the following proposition, which implies Lemma 9.

Proposition 1. *Suppose that in a full SMT for a set of n terminal points, there is a Steiner path $q_1, q_2, \cdots, q_{k+1}$, $k \geq 1$, such that all Steiner points on this path are of degree 3 and q_1 is incident with s_1, s_2; q_{k+1} is incident with s_{k+2}, s_{k+3}; and q_i is incident with s_{i+1}, $2 \leq i \leq k$, where s_i is either a terminal point or a Steiner point outside the Steiner path. Then we have the following:*

1. *If all edges each incident with some Steiner point on this Steiner path except for $e[q_1, s_2]$ are straight, then we can transform $e[q_1, s_2]$ by length ε to an edge incident at q_{k+1} without changing the total length of the tree and all other edges remain straight.*
2. *If in addition there is also a non-straight edge, $e[q_{k+1}, s_{k+2}]$, incident at q_{k+1}, then we can transform $e[q_1, s_2]$ by length ε to edge $e[q_{k+1}, s_{k+2}]$ without changing the total length of the tree and all other edges remain straight.*

Proof. We prove the proposition by induction on k. When $k = 1$, the proposition is exactly Lemma 8. Therefore the proposition is true for $k = 1$. Suppose it is true for all value less than k, $k \geq 2$. We will prove that it is also true for k.

Consider the first case where no non-straight edge is incident at q_{k+1}. Since we can transform $e[q_1, s_2]$ by length ε_1 to an edge incident at q_k, if the edge is $e[q_k, q_{k+1}]$, then we are done. Otherwise, let $e[q_k, s]$ denote the destination edge. Note that s may be q_{k-1} or s_{k+1}. Applying Lemma 8 to the set of four points $q_{k-1}, s_{k+1}, s_{k+2}$ and s_{k+3}, we can transform $e[q_k, s]$ by length ε_2 to an edge $e[q_{k+1}, t]$ with $t = s_{k+2}$ or s_{k+3}. If $\varepsilon_1 \cdot \varepsilon_2 > 0$, then $|\varepsilon_1| \geq |\varepsilon_2|$. In this case, we can transform $e[q_k, s]$ by length $\varepsilon_1 - \varepsilon_2$ backwards to $e[q_1, s_2]$. This can be done since the transforming process consists of only parallel sliding operations and henceforth is reversible. By doing so, the edge $e[q_k, s]$ becomes straight and the proposition follows with $\varepsilon = \varepsilon_2$.

When $\varepsilon_1 \cdot \varepsilon_2$ is negative, we may transform $e[q_k, s]$ by length $\varepsilon_1 - \varepsilon_2$ backwards to $e[q_1, s_2]$. It may happen that for some ε', $e[q_1, s_2]$ becomes straight. Then we have $|\varepsilon'| > |\varepsilon_1|$ and then transform exactly $-\varepsilon_2 + \varepsilon_1 - \varepsilon'$ from $e[q_k, s]$ backwards to $e[q_{k+1}, t]$. By doing so, $e[q_k, s]$ becomes straight and the proposition follows with $\varepsilon = \varepsilon_1 - \varepsilon'$.

Now consider the case in which there is also a non-straight edge incident at q_{k+1}. The proof is similar to the above case. Since we can transform $e[q_1, s_2]$ by length ε to an edge $e[q_k, s]$, we know that $s \neq q_{k+1}$ by Lemma 5. We then have a four-point configuration in which there are two non-straight edges. Applying Lemma 8, we can transform $e[q_k, s]$ by length ε' to edge $e[q_k, s_{k+2}]$ such that $\varepsilon \cdot \varepsilon' > 0$. This also implies that $e[q_k, s]$ and $e[q_{k+1}, s_{k+2}]$ lie in different sides of path q_k-q_{k+1}. Note that we can decrease either ε or ε' to make edge $e[q_k, s]$ straight. That is, equivalently, transform some length backwards to either $e[q_1, s_2]$ or $e[q_{k+1}, s_{k+2}]$. The proposition follows. □

As was mentioned in [8], the transformation process done in the above proposition consists of a series of parallel sliding operations. It follows that in the case there is a non-straight edge incident at each of q_1 and q_{k+1}, increasing the value $|\varepsilon|$ will either make a point outside of path coincide with its adjacent Steiner point on the path, or make one of the two non-straight edges become straight.

Lemma 10. *If there is no non-full SMT for a set P of n terminal points, and there exists an SMT containing a degree-4 Steiner point, then in this SMT all edges are straight.*

Proof. Let us first examine the neighbors of a degree-4 Steiner point. See Figure 4, q is adjacent to the degree-4 Steiner point q_0 and it is not a leaf in the SMT, that is, q is itself a Steiner point. It follows that there is no edge incident at q going in directions $\frac{3\pi}{4}$ and $\frac{5\pi}{4}$. Moreover, there is no edge going out of q in direction $\frac{\pi}{2}$ since otherwise we can slide $\overline{q_0q}$ along $e[q_0, q_1]$ in direction $\frac{\pi}{2}$ and then contradicts Lemma 4. For the same reason, there is no edge going out of q in direction $\frac{3\pi}{2}$. Therefore, q is a degree-3 Steiner point incident with other two straight edges, one goes in direction $\frac{\pi}{4}$ and the other goes in direction $\frac{7\pi}{4}$.

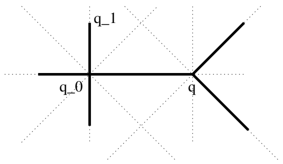


Fig. 4. A neighbor q of the degree-4 Steiner point q_0.

The lemma then follows from Proposition 1 by contradiction. We omit the detail proof here. □

Corollary 2. *In any full component of an SMT containing a degree-4 Steiner point, there is no non-straight edge.*

Proposition 1 and Lemma 10 imply the following.

Lemma 11. *If there is no non-full SMT for a set P of n terminal points, then there exists an SMT in which at most one edge is non-straight.* □

Corollary 3. *For any set of terminal points, there is an SMT such that there is at most one non-straight edge in each full component of it.*

Theorem 1. *For any given set P of n terminal points, $P = \{p_1, p_2, \cdots, p_n\}$, $n \geq 3$, there exists an SMT in which all the Steiner points are in $\mathcal{G}_{\lceil \frac{2n}{3} \rceil - 1}$, the $(\lceil \frac{2n}{3} \rceil - 1)^{st}$-generation grid formed by P.*

Proof. The proof is very long and it is done by careful analyses of the structures of the Steiner minimal trees for P. The main idea is that from Lemmas 4, 10 and 11, we may assume that any SMT T of P is full and contains only degree-3 Steiner points which lie on a path with the two leaf Steiner points not simultaneously in $\mathcal{G}_1$, i.e., the non-straight edge is incident to a leaf Steiner point. The argument then proceeds by induction on the number $n(\geq 6)$ of terminals (note that the theorem holds for $n = 3, 4, 5$). Comparing to the literatures, the technique used in [2,8,9] is to reduce an SMT problem for a set P of n points to an SMT problem for $n-1$ points in $\mathcal{G}_1(P)$; while the technique used in our paper is to reduce an SMT problem for a set P of n points to either an SMT problem for $n-2$ points in $\mathcal{G}_1(P)$; or an SMT problem for $n-k$ points in $\mathcal{G}_2(P)$ with $k=3$ or 4; or an SMT problem for $n-k$ points in $\mathcal{G}_3(P)$ with $k=5$ or 6. The details is omitted here (it is available from the authors). □

5 Concluding Remarks

In this paper, we have studied the Steiner tree problem in the λ_4-geometry plane. We have shown that for any set of n terminal points, there exists an SMT in which all Steiner points are in $\mathcal{G}_{\lceil \frac{2n}{3} \rceil -1}$, the $(\lceil \frac{2n}{3} \rceil - 1)^{st}$-generation grid formed by the given n terminal points. This improves the previous best known result [8,9] stating that for any set of n terminal points, there exists an SMT in which all Steiner points are in $\mathcal{G}_{n-2}$. We believe that the technique/idea in this paper may be used to obtain similar results in general λ_σ-geometry plane, and possibly better results in λ_3- and λ_4-geometry planes. It is also expected that the properties of the Steiner minimal trees proved in this paper may be used in the design of efficient approximation algorithms.

References

1. D.-Z. Du and F.K. Hwang, A proof of Gilbert-Pollak's conjecture on the Steiner ratio, *Algorithmica*, **7**(1992), 121–135. 327
2. D.Z. Du and F.K. Hwang, Reducing the Steiner problem in a normed space, *SIAM Journal on Computing*, **21**(1992), 1001–1007. 328, 329, 335
3. M.R. Garey, R.L. Graham and D.S. Johnson, The complexity of computing Steiner minimal trees, *SIAM Journal on Applied Mathematics*, **32**(1977), 835–859. 327
4. E.N. Gilbert and H.O. Pollak, Steiner minimal trees, *SIAM Journal on Applied Mathematics*, **16**(1968), 1–29. 327
5. M. Hanan, On Steiner's problem with rectilinear distance, *SIAM Journal on Applied Mathematics*, **14**(1966), 255–265. 328
6. F.K. Hwang, On Steiner minimal trees with rectilinear distance, *SIAM Journal on Applied Mathematics*, **30**(1976), 104–114. 328
7. F.K. Hwang, D. Richard, and P. Winter, *The Steiner tree problems*, Annals of Discrete Mathematics, **53**(1992), North-Holland. 327
8. D.T. Lee and C.-F. Shen, The Steiner minimal tree problem in the λ-geometry plane, in *Proceedings of 7th International Symposium on Algorithms and Computations (ISAAC'96)*, LNCS 1178, 1996, 247–255. 328, 329, 330, 333, 334, 335, 336
9. D.T. Lee, C.-F. Shen and C.-L. Ding, On Steiner tree problem with 45^o routing, in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS'95)*, 1995, 1680–1683. 328, 329, 330, 335, 336
10. Y.Y. Li, S.K. Cheung, K.S. Leung and C.K. Wong, On the Steiner tree problem in λ_3-metric, in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'97)*, Hong Kong, June 9-12, 1997, 1564–1567. 328
11. G.-H. Lin and G.L. Xue, Reducing the Steiner problem in an A_3-geometry plane, *manuscript*, March, 1998. Submitted for journal publication. Available at http://www.emba.uvm.edu/~xue/manuscripts. 328

12. M. Sarrafzadeh and C.K. Wong, Hierarchical Steiner tree construction in uniform orientations, *IEEE Transactions on Computer-Aided Design*, **11**(1992), 1095–1103. 328
13. P. Widmayer, Y.F. Wu and C.K. Wong, On some distance problems in fixed orientations, *SIAM Journal on Computing*, **16**(1987), 728–746. 328
14. G.Y. Yan, A. Albrecht, G.H.F. Young and C.K. Wong, The Steiner tree problem in orientation metrics, *Journal of Computer and System Sciences*, **55**(1997), 529–546. 328, 329

Approximation and Exact Algorithms for RNA Secondary Structure Prediction and Recognition of Stochastic Context-Free Languages

Tatsuya Akutsu

Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan
`takutsu@ims.u-tokyo.ac.jp`

Abstract. For a basic version (i.e., maximizing the number of base-pairs) of the RNA secondary structure prediction problem and the construction of a parse tree for a stochastic context-free language, $O(n^3)$ time algorithms were known. For both problems, this paper shows slightly improved $O(n^3(\log\log n)^{1/2}/(\log n)^{1/2})$ time exact algorithms. Moreover, this paper shows an $O(n^{2.776})$ time approximation algorithm for the former problem and an $O(n^{2.976}\log n)$ time approximation algorithm for the latter problem, each of which has a guaranteed approximation ratio $1-\epsilon$ for any fixed constant $\epsilon > 0$, where the absolute value of the logarithm of the probability is considered as an objective value in the latter problem. Several related results are shown too.

1 Introduction

RNA secondary structure prediction is an important problem in *computational biology*. This is a problem of, given an RNA sequence of length n, finding its correct secondary structure (an outerplanar graph like structure, see Fig. 1). Usually, it is modeled as a *free-energy minimization* problem [9,12,17], for which simple DP (*dynamic programming*) algorithms have been proposed [15,18]. The time complexities are $O(n^3)$ if we ignore the *destabilizing energy* due to *loop regions*, otherwise they are at least $O(n^4)$.

In a basic and simplest version, free-energy minimization of an RNA secondary structure is defined as a problem of *maximizing the number of complementary base pairs* (see Fig. 1) [9,17], which is denoted by $\mathcal{RNA}_0$ in this paper. Even for $\mathcal{RNA}_0$, only an $O(n^3)$ time simple DP algorithm had been known [9].

Although no further improvement had been done on *global free-energy minimization*, several important improvements have been done for finding *locally stabilizing substructures* in an RNA secondary structure [17]. An $O(n^3)$ time algorithm was developed for an arbitrary destabilizing energy function [16]. An $O(n^2)$ time algorithm was developed for a *linear* destabilizing energy function [6], and near $O(n^2)$ time algorithms were developed for a concave or convex destabilizing energy function [4,7].

On the other hand, *stochastic context-free grammars* (*SCFG*, in short) have been applied to RNA secondary structure prediction, in which a parse tree with the highest probability (an *optimal parse tree*) corresponds to an RNA secondary structure with the minimum free-energy (an *optimal RNA secondary structure*) [8]. However, to my knowledge, only an $O(n^3)$ time algorithm was known for the construction of an optimal parse tree for SCFG.

In this paper, we first show a slightly improved $O(n^3(\log\log n)^{1/2}/(\log n)^{1/2})$ time exact algorithm for the construction of an optimal parse tree for SCFG, which can also be applied to $\mathcal{RNA}_0$. This algorithm is a simple combination of Valiant's algorithm for *context-free recognition* [14] with a fast algorithm for *funny matrix multiplication* [5,10]. Note that funny matrix multiplication is, given $p \times q$ real matrix $X = (x_{ij})$ and $q \times r$ real matrix $Y = (y_{ij})$, to compute $p \times r$ matrix $Z = (z_{ij})$ such that $z_{ij} = \max_{1 \le k \le q}(x_{ik} + y_{kj})$. For several problems such as the all-pairs shortest path problem [2,10] and the maximum subarray problem [11], the fastest algorithms were obtained using fast funny matrix multiplication.

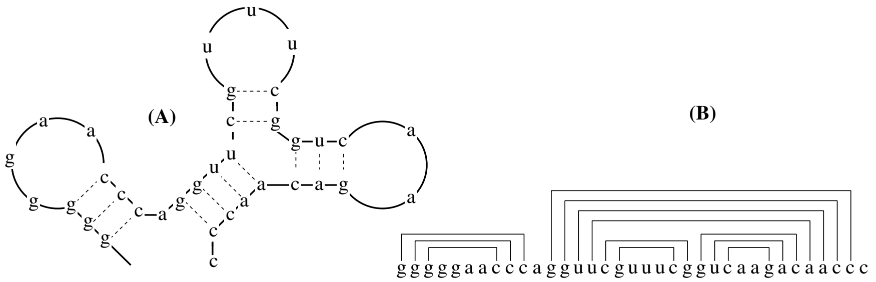

(A)
(B)
g g g g g a a c c c a g g u u c g u u u c g g u c a a g a c a a c c c

Fig. 1. Two representations of RNA secondary structure: (A) 'Clover leaf' representation similar to real structure; (B) Sequence is represented on the horizontal axis. In a basic version, secondary structure prediction is defined as a problem of maximizing the number of base pairs ((a,u), (g,c) pairs) which do not intersect each other.

Next, we show the main result of this paper: an $O(n^{2.776})$ time algorithm for $\mathcal{RNA}_0$ which always outputs, for any fixed constant $\epsilon > 0$, an RNA secondary structure with the score at least $1-\epsilon$ of the maximum, where the score denotes the number of complementary base pairs in $\mathcal{RNA}_0$. Although this algorithm is a PTAS (polynomial time approximation scheme), it is different from usual PTAS: the problem is not NP-hard but belongs to P; the time complexity does not depend on ϵ (ϵ converges to 0 as n grows). This algorithm is a combination of an approximation algorithm $\mathcal{A}_{approx}$ and an exact algorithm $\mathcal{A}_{exact}$, where $\mathcal{A}_{approx}$ is obtained by modifying the original $O(n^3)$ time DP algorithm for $\mathcal{RNA}_0$, and $\mathcal{A}_{exact}$ is obtained by combining Valiant's algorithm with fast funny matrix multiplication. Although Tamaki and Tokuyama developed an $o(n^3)$ time approximation algorithm for the maximum subarray problem based on fast funny matrix multiplication [11], their technique could not be applied to $\mathcal{RNA}_0$ and thus a new technique was introduced.

Then, we extend the technique used in $\mathcal{A}_{approx}$ for more practical versions of RNA secondary structure prediction. Since $\mathcal{A}_{approx}$ and their variants are very simple, they may be practical. We also modify the technique for the construction of an optimal parse tree for SCFG, and obtain an $O(n^{2.976} \log n)$ time approximation algorithm, which always outputs a parse tree with the score at least $1-\epsilon$ of the optimal, for any fixed $\epsilon > 0$, where we consider the absolute value of the logarithm of the probability as the score.

2 RNA Secondary Structure and SCFG

2.1 A Basic Version of RNA Secondary Structure Prediction

Let $A = a_1 a_2 \dots a_n$ be an *RNA sequence*. That is, A is a string over an alphabet $\Sigma = \{\mathtt{a}, \mathtt{u}, \mathtt{g}, \mathtt{c}\}$. A pair of residues (letters) (x, y) is called a (*complementary*) *base pair* if $\{x, y\} = \{\mathtt{a}, \mathtt{u}\}$ or $\{x, y\} = \{\mathtt{g}, \mathtt{c}\}$. Although we do not treat $\{\mathtt{g}, \mathtt{u}\}$ as a base pair, similar results hold for such a case. A set of pairs of indices $M = \{(i, j) |\ 1 \le i < j \le n,\ (a_i, a_j) \text{ is a base pair}\}$ is called an *RNA secondary structure* if no distinct pairs $(a_i, a_j), (a_h, a_k)$ in M satisfy $i \le h \le j \le k$ (see Fig. 1). The score of M is defined as the number of base pairs in M (i.e., $|M|$), and denoted by $score(M)$. Then, $\mathcal{RNA}_0$ is defined as follows: given an RNA sequence $A = a_1 a_2 \dots a_n$, to find an RNA secondary structure M with the maximum score. In $\mathcal{RNA}_0$, such a structure is also called an *optimal RNA secondary structure*, and denoted by $OPT_0(A)$.

It is well known that the score of $OPT_0(A)$ can be computed in $O(n^3)$ time using the following simple DP procedure (denoted by $\mathcal{DP}_0$):

$$S(i,j) \;=\; \max \begin{cases} S(i+1, j-1) + \mu(a_i, a_j), \\ \max_{i<k\le j}\{\, S(i, k-1) + S(k, j)\,\}, \end{cases}$$

where we let $S(i, j) = 0$ for all $i \ge j$, and $\mu(x, y) = 1$ if (x, y) is a base pair, otherwise $\mu(x, y) = 0$. Note that the score of $OPT_0(A)$ is given by $S(1, n)$. $OPT_0(A)$ can also be obtained in $O(n^3)$ time using the *traceback* technique [17]. Similarly, we only describe the procedures for computing scores or free-energies in this paper, *all of which can be modified for computing secondary structures or parse trees without increasing the orders of the time complexities.*

2.2 SCFG and Its Relationship with $\mathcal{RNA}_0$

A *stochastic context free-grammar* (SCFG) is a context-free grammar in which every production rule has an associated probability value. We denote the associated probability for a production $X \rightarrow \alpha$ by $P(X \rightarrow \alpha)$. Usually, $\sum P(X \rightarrow \alpha) = 1$ should hold for each non-terminal symbol X, where the sum is taken over all rules whose left hand sides are X.

The *probability of a parse tree* is defined by the product of the probabilities of the productions used to generate the sequence. The *probability of a sequence s* is the sum of probabilities over all possible parse trees that could generate s. An *optimal parse tree* is a parse tree with the highest probability.

Several papers pointed out a relationship between RNA secondary structure and SCFG [8,13]. Based on them, we can associate the following context-free grammar with $\mathcal{RNA}_0$:

	score of rule	$score(X)$
$X \longrightarrow \epsilon$	0	0
$X \longrightarrow \mathtt{a}$	0	0
$X \longrightarrow \mathtt{u}$	0	0
$X \longrightarrow \mathtt{g}$	0	0
$X \longrightarrow \mathtt{c}$	0	0
$X \longrightarrow YZ$	0	$score(Y) + score(Z)$
$X \longrightarrow \mathtt{a}Y\mathtt{u}$	1	$score(Y) + 1$
$X \longrightarrow \mathtt{u}Y\mathtt{a}$	1	$score(Y) + 1$
$X \longrightarrow \mathtt{g}Y\mathtt{c}$	1	$score(Y) + 1$
$X \longrightarrow \mathtt{c}Y\mathtt{g}$	1	$score(Y) + 1$

where a score is associated for each production rule instead of a probability, X, Y, Z denote the same non-terminal symbol, and $score(X)$ denotes the score of a node in a parse tree corresponding to the production rule. In this case, *a parse tree whose root has the maximum score* (among all parse trees) corresponds to *an optimal secondary structure* in $\mathcal{RNA}_0$ [8]. Note that, in this case, the score of a parse tree is *not the product of the probabilities*, *but the sum of the scores* of productions used to generate the sequence. However, a parse tree with the highest probability is equal to a parse tree with the maximum score if we assign score $\log p$ (≤ 0) to each production rule having probability p.

3 Exact Algorithms

Valiant developed an $O(n^\omega)$ time algorithm for context-free recognition [14] using a fast boolean matrix multiplication algorithm, where $O(N^\omega)$ deonotes the time complexity of the current best algorithms ($\omega = 2.376$ [3]) for both the boolean matrix multiplication and the usual (real) matrix multiplication for $N \times N$ matrices. Note that, following to the standard convention, we consider a fixed grammar and the size of the grammar is assumed to be a constant. Moreover, we assume without loss of generality that a grammar is given in the *Chomsky normal form* as in [14]. Valiant's algorithm computes an $n \times n$ matrix $X = (x_{ij})$ for a sequence $s = s_1 s_2 \ldots s_n$ such that $x_{ij} = 1$ if there is a parse tree for $s_i s_{i+1} \ldots s_{j-1}$, otherwise $x_{ij} = 0$.

Modifying Valiant's algorithm, we can obtain the following $o(n^3)$ time algorithms.

Theorem 1. *In SCFG, the probability of a given sequence s can be computed in $O(n^{\omega})$ time.*

Proof. We modify Valiant's algorithm so that x_{ij} represents the probability of a subsequence $s_i s_{i+1} \ldots s_{j-1}$ in SCFG. For that purpose, we replace the boolean matrix multiplication in Valiant's algorithm with the real matrix multiplication (along with other miscellaneous modifications). Since the real matrix multiplication for $N \times N$ matrices can be done in $O(N^{\omega})$ time, the time complexity remains $O(n^{\omega})$. □

Theorem 2. *In SCFG, an optimal parse tree can be computed in $O(n^3 (\log\log n)^{1/2}/(\log n)^{1/2})$ time, where we assume that the logarithm of the probability associated with each production rule can be expressed with $O(\log n)$ bits.*

Proof. In this case, we assign $\log p$ (≤ 0) as a score to each production rule with probability p, and we modify Valiant's algorithm so that x_{ij} denotes the score of an optimal parse tree for subsequence $s_i s_{i+1} \ldots s_{j-1}$ (if there is no parse tree, the score is set to $-\infty$). Then, such matrix X can be computed by replacing the boolean matrix multiplication with the funny matrix multiplication (along with miscellaneous modifications). Since funny matrix multiplication for $N \times N$ matrices takes $O(N^3(\log\log N)^{1/2}/(\log N)^{1/2})$ time (even if negative entries are included) [10], the time complexity becomes $O(n^3(\log\log n)^{1/2}/(\log n)^{1/2})$. Recall that, once a matrix X is obtained, an optimal parse tree can be obtained using the traceback technique. □

Since an optimal secondary structure in $\mathcal{RNA}_0$ corresponds to an optimal parse tree and the score for each rule is represented with $O(1)$ bits, we have:

Corollary 1. *$\mathcal{RNA}_0$ (i.e, finding an RNA secondary structure with the maximum number of base pairs) can be solved in $O(n^3(\log\log n)^{1/2}/(\log n)^{1/2})$ time.*

4 Approximation Algorithms for $\mathcal{RNA}_0$

4.1 Algorithm with a Constant Approximation Ratio

Here, we give a simple algorithm for $\mathcal{RNA}_0$ which always outputs an RNA secondary structure with the score at least 1/2 of the maximum. Although this algorithm is not essential for obtaining an $1-\epsilon$ approximation algorithm for $\mathcal{RNA}_0$, it is useful to reduce the time complexity of the $1-\epsilon$ approximation algorithm with a constant factor.

Proposition 1. *Suppose that an RNA sequence A consists of letters of* `a` *and* `u`*, and let* $\#$`a` *and* $\#$`u` *be the numbers of occurrences of* `a` *and* `u` *in A respectively. Then, the score of $OPT_0(A)$ is equal to $\min\{\#\mathtt{a}, \#\mathtt{u}\}$. Moreover, $OPT_0(A)$ can be computed in linear time.*

Proof. Using the following procedure, we can compute $OPT_0(A)$ in linear time, which consists of $\min\{\#\mathtt{a}, \#\mathtt{u}\}$ base pairs.

```
Let $S$ be an empty stack;
for $i = 1$ to $n$ do
  if $S$ is empty or $(a_i, top(S))$ is not a base pair then $push(a_i, S)$
  else begin Output $(a_i, top(S))$ as a base pair; $pop(S)$ end
```
□

Let $A(\mathtt{a}, \mathtt{u})$ (resp. $A(\mathtt{c}, \mathtt{g})$) be the subsequence of A consisting of letters of $\mathtt{a}$ and $\mathtt{u}$ (resp. $\mathtt{c}$ and $\mathtt{g}$). Then, $score(OPT_0(A))$ is at most the sum of $score(OPT_0(A(\mathtt{a}, \mathtt{u})))$ and $score(OPT_0(A(\mathtt{c}, \mathtt{g})))$. Choosing the better one, we have:

Theorem 3. *For $\mathcal{RNA}_0$, an RNA secondary structure with the score at least 1/2 of the maximum can be computed in linear time.*

4.2 $1 - \epsilon$ Approximation Algorithm

The $1-\epsilon$ approximation algorithm is a combination of an exact algorithm $\mathcal{A}_{exact}$ and an approximation algorithm $\mathcal{A}_{approx}$: $\mathcal{A}_{exact}$ is used when $score(OPT_0(A))$ is small (presizely, $score(OPT_0(A)) = O(n^\gamma)$ where γ is a constant to be determined later), otherwise $\mathcal{A}_{approx}$ is used.

First, we describe $\mathcal{A}_{exact}$. Recall that funny matrix multiplication for $N \times N$ integer matrices whose maximum absolute value of the entries is bounded by Q can be done in $O(Q(\log Q)N^\omega)$ time [2,11]. Using this in the algorithm in Sect. 3, we obtain $\mathcal{A}_{exact}$.

Lemma 1. *$\mathcal{A}_{exact}$ computes $OPT_0(A)$ in $O(Q(\log Q)n^\omega)$ time if $score(OPT_0(A)) \le Q$.*

Proof. The maximum absolute value of entries in matrices appearing in the execution of $\mathcal{A}_{exact}$ is bounded by $score(OPT_0(A))$. Therefore, each funny matrix multiplication for $N \times N$ matrices can be done in $O(Q(\log Q)N^\omega)$ time. It is straight-forward to see that the total time complexity is $O(Q(\log Q)n^\omega)$. □

Next, $\mathcal{A}_{approx}$ is obtained by modifying the original $O(n^3)$ time DP procedure $\mathcal{DP}_0$ for $\mathcal{RNA}_0$. Note that $S(i,j)$ in $\mathcal{DP}_0$ is equal to $score(OPT_0(a_i a_{i+1} \dots a_j))$.

Lemma 2. $|S(i,j) - S(i+h, j+k)| \le |h| + |k|$.

Proof. From the definition of $\mathcal{RNA}_0$, both $|S(i,j) - S(i,j+1)| \le 1$ and $|S(i,j) - S(i+1,j)| \le 1$ hold. □

In $\mathcal{A}_{approx}$, we do not compute $\max_{i<k\le j}\{\, S(i,k-1) + S(k,j) \,\}$ exactly. Instead, we compute the maximum of $S(i,k-1) + S(k,j)$ for $O(n^\alpha + n^{1-\beta})$ values of k's (see Fig. 2), where α and β ($0 < \alpha, \beta < 1$) are appropriate constants to be determined later.

We define a sequence of indices $f_i^+(h)$ and $f_j^-(h)$ for $h = 0, 1, 2, \cdots$ by

$$\begin{aligned} f_i^+(0) &= i + \lceil n^\alpha \rceil, & f_j^-(0) &= j - \lceil n^\alpha \rceil, \\ f_i^+(h+1) &= f_i^+(h) + \lceil (f_i^+(h) - i)^\beta \rceil, & f_j^-(h+1) &= f_j^-(h) - \lceil (j - f_j^-(h))^\beta \rceil. \end{aligned}$$

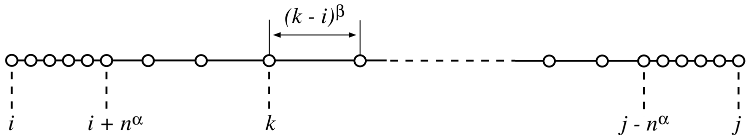


Fig. 2. In $\mathcal{A}_{approx}$, $\max_k S(i, k-1) + S(k, j)$ is computed not for all k, but for $O(n^{\alpha} + n^{1-\beta})$ values of k's, where such k's are represented by white circles in this figure.

Next, we define $\mathcal{I}(i, j)$ by

$$\mathcal{I}(i,j) = \{\ k \mid i < k \le n^{\alpha} \text{ or } j - n^{\alpha} \le k \le j\ \} \bigcup \{\ f_i^+(h) \mid f_i^+(h) \le (i+j)/2\ \} \bigcup \{\ f_j^-(h) \mid f_j^-(h) \ge (i+j)/2\ \}.$$

Then, $\mathcal{A}_{approx}$ is expressed by the following DP procedure:

$$S'(i,j) \;=\; \max \begin{cases} S'(i+1, j-1) + \mu(a_i, a_j), \\ \max_{k \in \mathcal{I}(i,j)} \{\, S'(i, k-1) + S'(k, j)\,\}, \end{cases}$$

where we let $S'(i, j) = 0$ for $i \ge j$.

Lemma 3. *$\mathcal{A}_{approx}$ works in $O(n^{2+\alpha} + n^{3-\beta})$ time.*

Proof. Since $j - i \le n$, the size of $\mathcal{I}(i, j)$ is bounded by

$$|\mathcal{I}(i,j)| \;\le\; 2n^{\alpha} \;+\; 4\Big(\, \frac{\frac{n}{2}}{(\frac{n}{2})^{\beta}} + \frac{\frac{n}{4}}{(\frac{n}{4})^{\beta}} + \frac{\frac{n}{8}}{(\frac{n}{8})^{\beta}} + \;\cdots\;\Big) \;\le\; O(n^{\alpha} + n^{1-\beta}).$$

Since $\max_{k \in \mathcal{I}(i,j)} \{\, S'(i, k-1) + S'(k, j)\,\}$ is computed for $O(n^2)$ pairs (i, j), $\mathcal{A}_{approx}$ takes $O(n^{2+\alpha} + n^{3-\beta})$ time. □

Here, we define the *error* of an secondary structure M to $OPT_0(A)$ to be $score(OPT_0(A)) - score(M)$ (note that this value must be non-negative).

Lemma 4. *The error of an secondary structure M computed by $\mathcal{A}_{approx}$ is $O(n^{1+\alpha\beta-\alpha})$.*

Proof. Note that, for each (i, j), we define the *error* of $S'(i, j)$ (in $\mathcal{A}_{approx}$) to be $S(i, j) - S'(i, j)$. Here, we show that, for all i, j, the following inequality holds:

$$S(i,j) - S'(i,j) \;\le\; \max\{\, C \cdot m \cdot n^{\alpha\beta-\alpha} - C \cdot m^{\beta},\ 0\}$$

for some constant C, where we let $m = j - i$. We prove this inequality by the induction on m.

Case (i) $m \le n^{\alpha}$

In this case, the error is always 0 and thus the inequality holds.

Case (ii) $m > n^{\alpha}$

In this case, we assume that the inequality holds for all m' such that $m' < m$, and we consider the following recurrence in $\mathcal{A}_{approx}$:

$$S'(i,j) \;=\; \max_{k \in \mathcal{I}(i,j)} \{S'(i,k-1) + S'(k,j)\}.$$

Let k' be the integer maximizing $S'(i,k'-1) + S'(k',j)$ under the condition that $i < k' \leq j$, and let $k'' \in \mathcal{I}(i,j)$ be the integer maximizing $S'(i,k''-1) + S'(k'',j)$ under the condition that $k'' \in \mathcal{I}(i,j)$. From Lemma 2 and the definition of $\mathcal{I}(i,j)$, it is seen that $S'(i,k'-1)+S'(k',j)-S'(i,k''-1)-S'(k'',j)$ is $O(h^{\beta})$, where $h = \min(k''-1-i, j-k'')$. Then, the error of $S'(i,j)$ is bounded by

$$C \cdot h \cdot n^{\alpha\beta-\alpha} \;-\; C \cdot h^{\beta} \;+\; C \cdot (m-h) \cdot n^{\alpha\beta-\alpha} \;-\; C \cdot (m-h)^{\beta} \;+\; D \cdot h^{\beta}$$

where D is an appropriate constant, and we assume without loss of generality that $h > n^{\alpha}$. It is not difficult to verify that this value is at most $C \cdot m \cdot n^{\alpha\beta-\alpha} - C \cdot m^{\beta}$ for $C \gg D$. □

Theorem 4. *For $\mathcal{RNA}_0$, an RNA secondary structure with the score at least $1-\epsilon$ of the maximum can be computed in $O(n^{2.776})$ time, where ϵ is any fixed positive number.*

Proof. First, we estimate $score(OPT_0(A))$ using the algorithm described in Sect. 4.1. If the estimated value is at most n^{γ}, an optimal structure is computed using $\mathcal{A}_{exact}$, otherwise an approximate structure is computed using $\mathcal{A}_{approx}$. Then, from Lemma 3, the time complexity is $O(n^{\gamma+\omega}\log n + n^{2+\alpha} + n^{3-\beta})$.

From Lemma 4, the ratio of the score of an approximate solution $\mathcal{A}_{approx}$ to the optimal score is $\dfrac{score(OPT_0(A)) - O(n^{1+\alpha\beta-\alpha})}{score(OPT_0(A))}$, which is greater than $1-\epsilon$ for any fixed $\epsilon > 0$ if $1+\alpha\beta-\alpha < \gamma$ and n is sufficiently large.

Here, we let $\alpha = 0.776$, $\beta = 0.224$, $\gamma = 0.398$ and $\omega = 2.376$. Then, $1+\alpha\beta-\alpha < \gamma$ is satisfied and the theorem follows. □

5 Approximation Algorithms for More Practical Cases

Although the above algorithms are not practical, the technique developed for $\mathcal{A}_{approx}$ can be applied to more practical versions of RNA secondary structure prediction. For example, we can obtain the followings (details are omitted).

Energy function defined for adjacent base pairs
In $\mathcal{RNA}_0$, energy function is defined for each base pair, whereas energy functions defined for adjacent base pairs are widely used [12,13]. For these functions, we can obtain an $O(n^{2.776})$ time $1-\epsilon$ approximation algorithm.

Destabilizing energy function
We did not consider free-energy for unpaired residues so far. However, such residues are also important determinants of RNA stability, and several energy functions are proposed for unpaired residues [9,12,17]. For most types

of destabilizing energy functions, we can obtain an $O(n^{2+\alpha} + n^{3-\beta})$ time algorithm with error at most $O(n^{1+\alpha\beta-\alpha})$.

Pseudoknots

Although *pseudoknots* (special kinds of substructures) are taken into account in a few algorithms, pseudoknots appear in several important RNA's [13]. For a basic version (i.e., maximizing the number of base pairs) of RNA secondary structure prediction with *simple* pseudoknots, $O(n^4)$ time algorithms were proposed [1,13]. For this problem, we can obtain an $O(n^{3.5})$ time algorithm with error at most $O(n^{0.75})$.

6 Approximation Algorithm for SCFG

Since there is a close relationship between $\mathcal{RNA}_0$ and SCFG, it is natural to try to extend Theorem 4 for SCFG. Unfortunately, Lemma 2 or similar property does not hold for general SCFG and thus we can not directly extend Theorem 4. However, we can still compute approximate scores for SCFG in $O(n^{3-\delta})$ time for some δ by modifying Valiant's algorithm.

Since we consider a fixed grammar, we assume that the score for each production rule is bounded by a constant. Moreover, we assume that each score is represented with finite bits. Then, we can assume that the optimal score is $O(n)$.

In Sect. 3, we used funny matrix multiplication for computing an optimal parse tree. Here, we use the following approximate funny matrix multiplication, where similar technique was used in [11]. We assume that the absolute value of each entry of $N \times N$ matrices A and B is bounded by $const \cdot N$. For a matrix $C = (c_{ij})$, $C' = (c'_{ij})$ denotes the matrix such that $c'_{ij} = \lfloor c_{ij}/\sqrt{N} \rfloor$. Since funny matrix multiplication for $N \times N$ integer matrices whose maximum absolute value of the entries is bounded by Q can be done in $O(Q(\log Q)N^{\omega})$ time, funny matrix multiplication for A' and B' can be done in $O(N^{\omega+0.5}\log N)$ time and the error for each entry of $A' \cdot B'$ (to $A \cdot B$) is at most $O(\sqrt{N})$. Moreover, we can remove the assumption on the maximum absolute value for B.

Theorem 5. *A parse tree for SCFG whose score is at least* $1-\epsilon$ *of the maximum can be computed in* $O(n^{2.976}\log n)$ *time, where* ϵ *is any fixed positive number.*

Proof. As in Theorem 4, we combine an approximation algorithm and an exact algorithm, where the former is obtained by modifying Valiant's algorithm using approximate funny matrix multiplication.

Recall that x_{ij} denotes the score of an optimal parse tree for subsequence $s_i s_{i+1} \dots s_{j-1}$ in Theorem 2. Here, we compute the exact score for an optimal parse tree for each subsequence with length at most n^{α} and we compute an approximate score for each subsequence with length more than n^{α}. Moreover, if s' is obtained by concatenating s^1 and s^2 (using some production rule), we will make the error due to this concatenation be at most $O(\sqrt{\min\{|s^1|,|s^2|\}})$ (we will make the error be 0 if $\min\{|s^1|,|s^2|\} < n^{\alpha}$), where $|x|$ denotes the length of sequence x. If we can do so, the total error is $O(n^{1+\alpha\beta-\alpha})$ as in Lemma 4. Note that we let $\alpha = 4/5, \beta = 1/2$ in this proof, and thus the total error is $O(n^{\frac{3}{5}})$.

Carefully checking (modified) Valiant's algorithm [14], we can see that, in each funny matrix multiplication for $N \times N$ matrices, at least one of input matrices P has the following property: each entry of P represents the score of an optimal parse tree for subsequence of length at most $2N$.

Using this property, we compute funny matrix multiplication $P \cdot Q$ approximately in the following way (we assume that P satisfies the above property). We divide P (resp. Q) into four $N/2 \times N/2$ submatrices. Then, we have

$$P \cdot Q = \begin{pmatrix} P_1 & P_2 \\ P_3 & P_4 \end{pmatrix} \cdot \begin{pmatrix} Q_1 & Q_2 \\ Q_3 & Q_4 \end{pmatrix} = \begin{pmatrix} P_1 \cdot Q_1 + P_2 \cdot Q_3 & P_1 \cdot Q_2 + P_2 \cdot Q_4 \\ P_3 \cdot Q_1 + P_4 \cdot Q_3 & P_3 \cdot Q_2 + P_4 \cdot Q_4 \end{pmatrix}$$

where $(\cdot, +)$ corresponds to $(+, \max)$. If $N < n^\alpha$, we compute funny matrix product $P \cdot Q$ exactly using $O(N^3)$ time. If $N \geq n^\alpha$, we compute funny matrix product $P \cdot Q$ approximately using the following recursive procedure. Since each entry in P_1, P_2, P_4 corresponds to a substring of length at least $N/2$, we compute $P'_i \cdot Q'_j$ instead of $P_i \cdot Q_j$ for $i \neq 3$. Since each entry in P_3 corresponds to a substring of length at most N, we apply the same procedure recursively to the approximate computation of $P_3 \cdot Q_1$ and $P_3 \cdot Q_2$. Then, the error due to the concatenation is at most $O(\sqrt{\min\{|s^1|, |s^2|\}})$. It is not difficult to see that the time for this approximate funny matrix multiplication is $O(N^{\omega+0.5} \log N)$. Thus, the total computation time for an approximate parse tree is $O(n^{\omega+0.5} \log n)$.

Since we compute an optimal parse tree simultaneously (assuming the score of an optimal parse tree is $O(n^{3/5})$) using $O(n^{\omega+\frac{3}{5}} \log n)$ time, the total computation time is $O(n^{\omega+\frac{3}{5}} \log n)$. □

References

1. Akutsu, T.: DP Algorithms for RNA Secondary Structure Prediction with Pseudo-knots. In: Genome Informatics 1997. Universal Academy Press, Tokyo(1997) 173–179 344
2. Alon, N., Galil, Z., Margalit, O.: On the Exponent of the All Pairs Shortest Path Problem. In: Proc. 32nd IEEE Symp. Found. Comput. Sci. (1991) 569–575 338, 342
3. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progression. J. Symbolic Computation **9** (1990) 251–280 340
4. Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G.F.: Sparse Dynamic Programming II: Convex and Concave Cost Functions. J. ACM **39** (1992) 546–567 337
5. Fredman, M.L.: New Bounds on the Complexity of the Shortest Path Problem. SIAM Journal on Computing **5** (1976) 83–89 338
6. Kanehisa, M., Goad, W.B.: Pattern Recognition in Nucleic Acid Sequences II: an Efficient Method for Finding Locally Stable Secondary Structures. Nucleic Acids Research **10** (1982) 265–277 337
7. Larmore, L.L., Schieber, B.: On-line Dynamic Programming with Applications to the Prediction of RNA Secondary Structure. J. Algorithms **12** (1991) 490–515 337
8. Sakakibara, Y., Brown, M., Hughey, E., Mian, I.S., Sjlölander, K., Underwood, R.C., Haussler, D.: Stochastic Context-Free Grammars for tRNA Modeling. Nucleic Acids Research **22** (1994) 5112–5120 338, 339, 340
9. Setubal, J., Meidanis, J.: Introduction to Computational Molecular Biology. PWS Pub. Co., Boston (1997) 337, 344

10. Takaoka, T.: A New Upper Bound on the Complexity of All Pairs Shortest Path Problem. Information Processing Letters **43** (1992) 195–199 338, 341
11. Tamaki, H., Tokuyama, T.: Algorithms for Maximum Subarray Problem Based on Matrix Multiplication. In: Proc. 9th ACM-SIAM Symp. Disc. Alg. (1998) 446–452 338, 342, 344
12. Turner, D.H., Sugimoto, N., Freier, S.M.: RNA Structure Prediction. Ann. Rev. Biophys. Biophys. Chem. **17** (1988) 167–192 337, 344
13. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Grammatically Modeling and Predicting RNA Secondary Structures. In: Proc. Genome Informatics Workshop VI. Universal Academy Press, Tokyo (1995) 67–76 339, 344
14. Valiant, L.G.: General Context-Free Recognition in Less Than Cubic Time. Journal of Computer and System Sciences **10** (1975) 308–315 338, 340, 345
15. Waterman, M.S., Smith, T.F.: RNA Secondary Structure: a Complete Mathematical Analysis. Math. Biosciences **41** (1978) 257–266 337
16. Waterman, M.S., Smith, T.F.: Rapid Dynamic Programming Algorithms for RNA Secondary Structure. Advances in Applied Mathematics **7** (1986) 455–464 337
17. Waterman, M.S.: Introduction to Computational Biology. Capman & Hall, London (1995) 337, 339, 344
18. Zuker, M., Stiegler, P.: Optimal Computer Folding for Large RNA Sequences Using Thermodynamics and Auxiliary Information. Nucleic Acids Research **9** (1981) 133–148 337

On the Multiple Gene Duplication Problem

Michael Fellows[1], Michael Hallett[2], and Ulrike Stege[2]

[1] Dep. of Computer Science, Univ. of Victoria, Victoria, B.C. Canada V8W 3P6
`mfellows@csr.uvic.ca`

[2] Comp. Biochemistry Research Group, ETH Zürich, CH-8092 Zürich, Switzerland
`{hallett,stege}@inf.ethz.ch`

Abstract. A fundamental problem in computational biology is the determination of the correct *species tree* for a set of taxa given a set of (possibly contradictory) *gene trees*. In recent literature, the Duplication/Loss model has received considerable attention. Here one measures the similarity/dissimilarity between a set of gene trees by counting the number of *paralogous gene duplications* and subsequent *gene losses* which need to be postulated in order to explain (in an evolutionarily meaningful way) how the gene trees could have arisen with respect to the species tree. Here we count the number of *multiple gene duplication events* (duplication events in the genome of the organism involving one or more genes) without regard to gene losses. Multiple Gene Duplication asks to find the species tree S which requires the fewest number of multiple gene duplication events to be postulated in order to explain a set of gene trees $G_1, G_2, \ldots, G_k$. We also examine the related problem which assumes the species tree S is known and asks to find the explanation for $G_1, G_2, \ldots, G_k$ requiring the fewest multiple gene duplications. Via a reduction to and from a combinatorial model we call the Ball and Trap Game, we show that the general form of this problem is *NP*-hard and various parameterized versions are hard for the complexity class $W[1]$. These results immediately imply that Multiple Gene Duplication is similarily hard. We prove that several parameterized variants are in *FPT*.

1 Introduction to the Model

A fundamental problem arising in computational biology is the determination of the (correct) evolutionary topology for a set of taxa given a set of (possibly contradictory) *gene trees*. A gene tree is a complete rooted binary tree formed over a family of homologous genes for a set of taxa. For various reasons, two or more gene trees for the same set of taxa may not always agree (see [2,5] amongst others). The question then arises of how to reconstruct the correct species tree for these taxa from the given gene trees. Several models have appeared in the literature including possibly the most famous MAST [7,11,12,13]. One such cost model which has received considerable attention of late is the Gene Duplication and Loss model introduced in [8] and discussed in [9,10,14]. The basic idea here is to measure the similarity/dissimilarity between a set of gene trees by counting the number of postulated *paralogous gene duplications* and subsequent *gene losses* required to explain (in an evolutionarily meaningful way) how the gene trees could have arisen with respect to the species tree. We use angle brackets $<,>$ to denote multisets. All trees in this paper are rooted and leaf labeled. Let $T = (V, E, L)$ be such a rooted tree where V is the vertex set, E is the edge set, and $L \subseteq V$ is the leaf label set. For a vertex $u \in V - L$, let T_u be the subtree of T rooted by u. Let $root(T)$ denote the root of T and, for any vertex $v \in V$, let $parent_T(v)$ be the parent of v in T, and for binary trees, let $left_T(v)$ be the left kid of v and $right_T(v)$ be the right kid of v. Where $L = \{1, 2, \ldots, n\}$ we call these leaf labeled trees either a *species* or a *gene* tree. Let $G = (V_G, E_G, L)$ be a gene tree and $S = (V_S, E_S, L')$, $L \subseteq L'$, be a species tree. We use a function $loc_{G,S} : V_G \rightarrow V_S$ to associate each vertex in G with a vertex in S. Furthermore, we use a function $event_{G,S} : V_G \rightarrow \{dup, spec\}$ to indicate whether the event in G corresponds to a duplication or speciation event. In [10], a function t_{dup} is given which returns the minimum number of *duplication events* necessary to rectify a gene tree with a species tree (here the algorithm from [9] is modified to ignore losses). Our function M below maps a gene tree G into a species tree S by defining functions $loc_{G,S}$ and $event_{G,S}$. It is the case that $t_{dup} = |\{u | u \in V_G - L, event_{G,S}(u) = dup\}|$.

$M(G, S)$: for each $u \in V_G - L$, $loc(u) = lca_S(u)$ and

$$event(u) = \begin{cases} spec \text{ if } loc(u') \neq loc(u), \text{for all } u' \text{ where } u' \text{ is a kid of } u \text{ in } G. \\ dup \text{ otherwise} \end{cases}$$

The following problem lies at the heart of the Duplication model:
Optimal Species Tree (Duplication Model):
Input: Set of gene trees $G_1, \ldots, G_k$. *Question:* Does there exist a species tree S with minimum duplication cost $\sum_{i=1}^{k} t_{dup}(G_i, S)$?

In [10], it is shown that the problem of finding the species tree S which minimizes the number of gene duplications is *NP*-complete (here the gene trees may contain leaf labels which appear more than once). When each gene tree may contain a leaf label at most once (a gene tree is formed over exactly one gene per taxa), the problem remains *NP*-complete and the parameterized (by k) version is hard for $W[1]$ (see [6]). A similar question to Optimal Species Tree arises if we ask for the species tree which implies the minimum number of *multiple gene duplications* for a given set of gene trees. A duplication event in the genome of an organism involves a stretch of DNA where one or more genes may reside. Previous models for rectifying gene trees with species trees considered a duplication event to effect one gene at a time. However, there is evidence that genomes of, for example, eukaryotic organisms, have been entirely duplicated one or more times or individual chromosomes have been duplicated multiple times. In either case, sets of genes were duplicated in one event creating a set of paralogous genes. Such paralogous duplications make finding the correct species topology especially difficult [5]. Consider a vertex u in a species tree S. Each gene tree G_i has some number of vertices (possibly zero) with $loc_{G,S}$ equal to u and $event_{G,S}$ equal to *dup*. Let $Dup = \{d_1, d_2, \ldots, d_c\}$ denote this set. We can partition the Dup into sets with the property that each set has at most one element from each G_i and so that these sets are maximal. One such set is termed a *multiple gene duplication* and it counts exactly one to the overall number of multiple gene duplications required to rectify the gene trees with respect to the species tree. The multiple gene duplication score for the vertex u is the total number of such partitions. By "moving" gene duplication events in G_i towards the root of S according to a set of rules, we can decrease the total number of multiple gene duplications required. Fig. 1 gives a concrete example.

Observation. *The duplication mapping function M given above (modified from [9,10]) provides an upper bound for the number of multiple gene duplications for a set of gene trees $G_1, G_2, \ldots, G_k$ and a species tree S.*

Let $G = (V_G, E_G, L)$ be a gene tree and $S = (V_S, E_S, L')$ a species tree, $L \subseteq L'$. Consider a vertex $u \in V_G$ such that $event_{G,S}(u) = dup$ and $u \neq root(G)$. Let $v = parent_G(u)$. The rules for moving duplication events towards the root of a species tree are as follows *Move 1: $event_{G,S}(v) = dup$.* We may move the duplication associated with u from $loc_{G,S}(u)$ to $loc_{G,S}(v)$ without creating any new duplications. Now $loc_{G,S}(u) = loc_{G,S}(v)$. *Move 2: $event_{G,S}(v) = spec$.* When moving the duplication associated with u from $loc_{G,S}(u)$ to $loc_{G,S}(v)$, we must change $event_{G,S}(v)$ to be dup. Now $loc_{G,S}(u) = loc_{G,S}(v)$.

Definition 1. *Given a gene tree G, a species tree S, and the functions $loc_{G,S}$ and $event_{G,S}$ mapping G into S, we say that S* receives *G, if the configuration given by $loc_{G,S}$ and $event_{G,S}$ can be reached by a series of moves starting from the initial configuration obtained by applying $M(G, S)$.*

The Multiple Gene Duplication problem can now be stated as follows:
Multiple Gene Duplication I
Input: Set of gene trees $G_1, \ldots, G_k$, integer c. *Question:* Do there exist a species tree S and functions $loc_{G_i,S}$, $event_{G_i,S}$, for $1 \leq i \leq k$, s.t. S receives $G_1, \ldots, G_k$ with at most c multiple gene duplications?

We state an easier version of Multiple Gene Duplication:
Multiple Gene Duplication Problem II (GDII)
Input: Set of gene trees $G_1, \ldots, G_k$, a species tree S, integer c. *Question:* Do there exist functions $loc_{G_i,S}$, $event_{G_i,S}$, for $1 \leq i \leq k$, s.t. S receives $G_1, \ldots G_k$ with at most c multiple gene duplications?

Via a reduction to and from a combinatorial problem called the Ball and Trap Game, we show $W[1]$-hardness and *NP*-completeness for GDII. We also show the problem is fixed-parameter tractable when various restrictions are placed on the number of gene trees and the number of the gene duplications. For an introduction to parameterized complexity we refer readers to [3,4].

2 The Ball and Trap Game

The Ball and Trap Game is played on a rooted labeled tree $T = (V, E, L)$ decorated with a set of traps D and a set of balls B. Every ball and trap has a color associated with it; this is given by the functions $c_B : B \to [1 : k]$ and $c_D : D \to [1 : k]$ respectively. The balls and traps are initially associated with internal vertices of T via the attaching functions $l_B : B \to V - L$ and $l_D : D \to V - L$. Each ball $b \in B$ of color $c_B(b)$ is labeled with a (possible empty) subset $R_b \subseteq D$ of

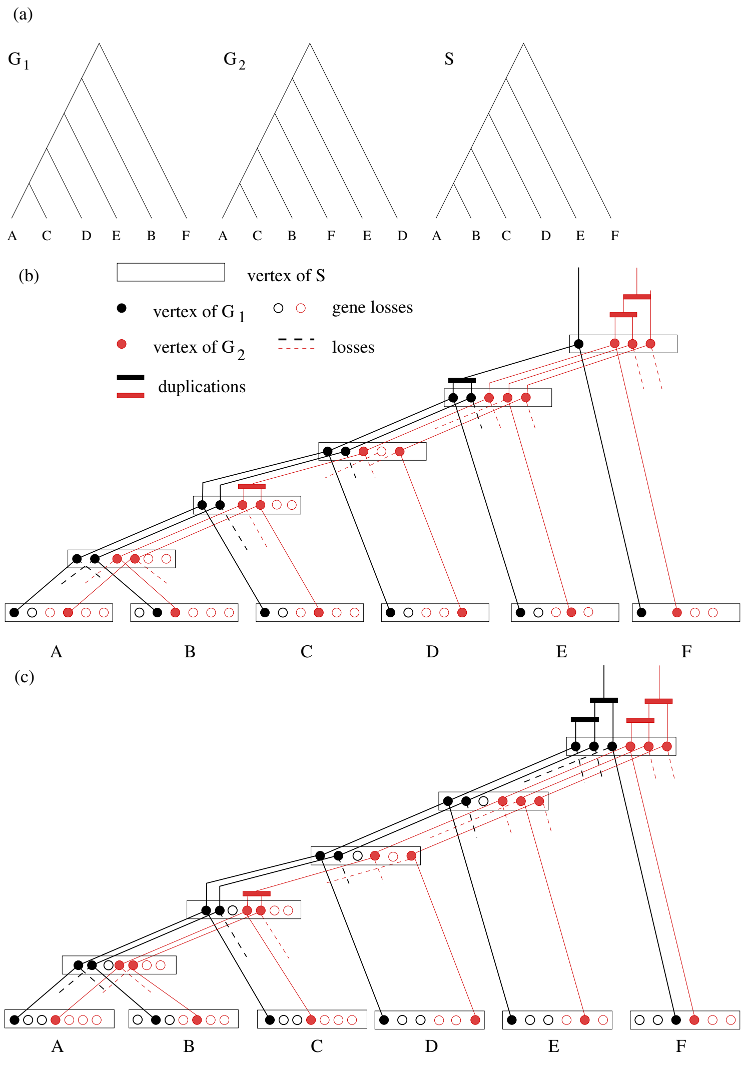


Fig. 1. (a) Two gene trees (G_1, G_2) and a proposed species tree S. (b) The species tree S has G_1 and G_2 embedded inside of it according to the standard Duplication and Loss mapping function M. Note that G_1 causes one duplication (vertex $ABCDE$) whilst G_2 causes 3 duplications (vertices ABC, $ABCDEF$, and again $ABCDEF$). The score according to the Multiple Gene Duplication model is 4. (c) After moving the gene duplication of G_1 located at vertex $ABCDE$ to the root of the species tree S, two additional gene duplications for G_1 need to be postulated. Nevertheless, the score according to the Multiple Gene Duplication model is now 3. Note that it is not beneficial to move the gene duplication from G_2 located at ABC towards the root. The two gene duplications located initially at the root from G_2 cannot move upwards.

traps. For each ball b every trap $d \in R_b$ is of the color $c_D(d) = c_B(b)$. A ball with a given set of traps may occur many times in the tree (i.e. for $b, b' \in B$ $R_b = R'_b$ and $c_B(b) = c_B(b')$ but $l_B(b) \neq l_B(b')$ is possible). Also, a vertex in the tree can be decorated with many different balls and traps.

A game consists of some number of moves, after which the score is calculated. The rules of the game are as follows: 1. Balls and traps are initially placed at internal vertices of T according to l_B and l_D. 2. Balls may not move down the tree. They may either stay in the same place or move upwards following the topology of T. In each turn, a ball b on a vertex v can be moved to the *parent*(v). 3. We say that a trap $d \in D$ is *dangerous* for a ball $b \in B$ if $d \in R_b$. A ball b *sets off* a trap if the ball is placed at the vertex of a trap dangerous for b. 4. When a trap d is *set off* by a ball b, it is removed from the game and replaced by two new balls $b_{new}, b_{new'}$ s.t. (a) $c_B(b_{new}) = c_B(b_{new'}) = c_B(b)$, (b) $l_B(b_{new}) = l_B(b_{new'}) = l_D(d)$, (c) $R_{b_{new}} = R_{b_{new'}} = R_B - d$ and $R_b = R_b - d$, and (d) $R_{b'} = R_{b'} - d$, for all $b' \in B$. The goal of the game is to minimize the score of tree T which is defined by $s_{max}(v) = \sum_{v \in V(T)} \max\{s(1, v), ..., s(k, v)\}$ where $s(c, v)$ denotes the number of balls of color c at vertex v in T.

Ball and Trap Game (Optimization)
Input: A rooted labeled tree T, a set of *balls* B, a set of *traps* D, two coloring functions $c_B : B \rightarrow [1 : k]$ and $c_D : D \rightarrow [1 : k]$, two initial location functions $l_B : B \rightarrow V_T - L$, $l_D : D \rightarrow V_T - L$, and for each ball $b \in B$ a set $R_b \subseteq D$ where for each $d \in R_b$ $c_D(d) = c_B(b)$.
A Round: Each round of the game consists of the player moving any number of same colored balls up the tree or deciding not to move any balls (halting move).
Output: The location function $l'(B)$ generated according to the above rules which minimizes $\Sigma_{\forall v \in V(T)} s_{max}(v)$.

The input is measured as follows: n denotes the size of T, k denotes the number of colors, r denotes the number of traps, and there are at most m balls on any vertex of T in the initial configuration. The above defined game leads to the following decision variant of the problem:

Ball and Trap (BT) - Decision
Input: A rooted tree T decorated with traps D and balls B in the manner described above, and a positive integer t. *Question:* Can the Ball and Trap Game be played on T to achieve a score of at most t?

Theorem 1. Multiple Gene Duplication II *reduces to* BT (Decision).

Proof(sketch). We construct an instance $I' \in$BT from an instance of $I \in$GDII. Let T equal the species tree S, $t = c$, and the number of colors k' of I' be the number of gene trees k from I. Each color corresponds to one of the input gene trees. Apply $M(G_i, S)$ and consider the functions $loc_{G_i,S}$ and $event_{G_i,S}$, for $1 \leq i \leq k$. We create a ball b with $c_B(b) = i$ for every vertex $u \in V_G$ s.t. $event_{G_i,S}(u) = dup$. Let $l_B(u) = loc_{G_i,S}(u)$. If $loc_{G_i,S}(u) \neq root(S)$, then let $Dup = \{d | d \in V_{G_i}, d$ is an ancestor of u, $event_{G_i,S}(d) = spec\}$. For each $d \in Dup$ we create a trap d' and let $l_D(d') = loc_{G_i,S}(d)$ in T and $c_D(d') = i$. Place d in R_b. The proof that $I' \in$BT$_{\text{``}yes''}$ if and only if $I \in$GDII$_{\text{``}yes''}$ is straightforward. One need only verify that the legal moves for a ball in the Ball and

Trap Game correspond to the legal moves for a duplication event for Multiple Gene Duplication. The only slightly tricky situation arises when there is a series $u_1, u_2, \ldots, u_q \in V_{G_i}$ s.t. $event_{G_i,S}(u_p) = dup$, $loc_{G_i,S} = x$ and u_p is a director descendant of u_{p+1} in G_i, for all $1 \leq p < q$. In Multiple Gene Duplication, one must first move duplication event p upwards before moving duplication events $1, \ldots, p-1$. When the ball corresponding to duplication event u_p is moved upwards to the same level as the ball corresponding to duplication event $u_{p'}$, $p < p'$, the traps dangerous for these two balls are equivalent.

3 Easy and Hard Parameterizations of the Ball and Trap Game

We consider the following parameterizations of Ball and Trap.
Ball and Trap:
Input: A rooted tree T decorated with traps D and balls B in the manner described above and integer t.
Parameters: $k = 2$, for each $b \in B$ let $|R_b| \leq 2$, number of traps r. (Version I)
Parameters: k, r, m, t. (Version II)
Parameters: k, r. (Version III)
Question: Can the Ball and Trap Game be played on T to achieve a score of at most t?

3.1 Easy Parameterizations

We can use finite-state dynamic programming on trees (equivalently, finite-state recognition of trees vertex labeled from a finite set of labels) in order to prove the following fixed-parameter tractability result.

Theorem 2. *For every fixed set of parameter values* (k, r, m, t)*, the problem* Ball and Trap II *can be solved in time linear in the size of the tree.*

Proof. Using the methods of [1], we can represent an input tree T as a labeled binary tree (even though T may not be binary), where the labels (which we will refer to as colors) indicate both the structure of T and the adornments of the vertices of T with balls and traps. The colored binary tree that represents an input tree T is called a *parse tree* for T. In this representation of T we can assume that the total number of balls of any given color on T is bounded by t, since otherwise T would be a "No" instance. We argue that there is a finite state tree automaton that recognizes precisely those labeled binary trees that represent "Yes" instances of the problem. Our argument is based on the "method of test sets" of [1,4]. Since there are at most $k2^r$ types of balls, and since each vertex may have m balls, $2^r(k2^r)^m$ colors suffice. The input trees are rooted, and we may assume that the parse tree for a given input tree T is rooted compatibly (i.e., at the same vertex). We use the following parsing operators: (1) the unary operator $\otimes_c$, for each color c, that has the effect of adding a single edge from the root to a new root colored c, and (2) the binary $\oplus$ operator (defined only for trees having the same color root) that takes as arguments two trees T_1 and T_2

and returns the tree $T = T_1 \oplus T_2$ obtained by identifying the root vertices of T_1 and T_2. We describe a set $\mathcal{T}$ of *tests*, each of which is a predicate $p(T)$ about a decorated tree T. We say that T *passes* the test if $p(T) = \texttt{true}$. Decorated trees T and T' are *equivalent*, denoted $T \sim T'$, if they: (1) have roots of the same color (i.e., that are decorated with traps and balls in the same way), and (2) pass the same set of tests in $\mathcal{T}$, i.e., if $\{p \in \mathcal{T} : p(T) = \texttt{true}\} = \{p \in \mathcal{T} : p(T') = \texttt{true}\}$ A test in $\mathcal{T}$ is specified by the following items: (1) A positive integer $t' \leq t$. (2) A length k vector $S = (s_1, ..., s_k)$ of non-negative integers, where each s_i is at most t. (3) The statement: "It is possible to play the Ball and Trap Game on T in such a way that at the end of play: • The total score on the internal vertices of T is at most t'. • The set of balls on the root of T consists of s_1 balls of color 1, s_2 balls of color $2, \ldots,$ and s_k balls of color k? To conclude the theorem it is enough to establish the following three claims. *Claim 1.* The equivalence relation $\sim$ has a finite number of equivalence classes. *Claim 2.* If $T_1 \sim T_2$ and T_1 is a *Yes*-instance for the problem, then T_2 is a *Yes*-instance also. *Claim 3.* The equivalence relation $\sim$ is a congruence with respect to the two parsing operators $\otimes_c$ and $\oplus$ for trees.

Theorem 3. Ball and Trap III *can be solved in time* n^c *where* $c = O((k2^r)^m)$.

Proof(sketch). We use leaf-to-root dynamic programming. At each vertex u of the tree T we calculate a table of pairs (c, S) where S is a set of balls to be passed upwards from u, and c is the minimum total score that can be achieved in the subtree T_u rooted at u assuming that the balls of S are passed upwards. It is easy to calculate this information for a vertex u from the information for the children of u. The best score that can be achieved for T is the value c at the root for $S = \emptyset$.

3.2 *W*-hardness of the Ball and Trap Game

We prove the $W[1]$-hardness of Ball and Trap I for parameter r by means of a polynomial-time parameterized reduction from Clique (Clique is proven complete for $W[1]$ in [3]). As an intermediate step we prove that the following parameterized problem is hard for $W[1]$.
k, r-Small Union
Input: A family $\mathcal{F}$ of subsets of $\{1, ..., n\}$, and positive integers r and k. *Parameter:* (r, k) *Question:* Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \geq r$ s.t. the union of the sets in $\mathcal{F}'$ has cardinality at most k?

Lemma 1. Small Union *is* NP*-complete and hard for* $W[1]$.

Theorem 4. Ball and Trap *is* NP*-complete, and* Version I *is hard for* $W[1]$.

Proof. Ball and Trap is well-defined for non-binary trees, and so we describe how Small Union can be reduced to Ball and Trap I. Let $(\mathcal{F}, r, k)$ be an instance of Small Union. We can assume, by the reduction described above, that $\mathcal{F}$ consists of 2-element sets. In order to describe the reduction we must describe a tree T with decorations, and the target value t for the game on T. The tree T is just a star of degree $n = |\mathcal{F}|$ (with the root being the central vertex). Each leaf of T is associated with an element of $\mathcal{F}$. The two colors are *red* and *blue*. There are n red traps $\tau_1, ..., \tau_n$. Each leaf is decorated with a single red ball labeled with the set of traps $\{\tau_u, \tau_v\}$ for the associated ("edge") set $\{u, v\}$. The root is decorated with $k + r$ blue balls, each labeled with the empty set of traps. The root is also decorated with all the n red traps. We set $t = (n - r) + (k + r) = n + k$. The basic idea for the correctness argument can be described as follows. Initially, the score is $n + k + r$. The only possible move is to move a red ball from a leaf to the root. If r balls can be moved up to the root from the leaves, with the r balls chosen so that the union of their trap label sets has cardinality k, then the result is a total of $k + r$ red balls at the root (where there are $k+r$ blue balls, so the cost of the root in the final score remains $k + r$). Thus the score at the end is t. Conversely, if a score of t is achieved by a game g, then necessarily at least r red balls must be moved up from the leaves. Let g' denote the truncated game consisting of the first r moves. There are two possibilities: 1. g' also achieves a score of at most t, and 2. the score for the game g' is greater than t. In case 1, exactly r red balls are moved to the root and consequently the score for the root vertex is at most $k + r$, which implies that the union of the trap label sets for the balls moved up has cardinality at most k. This implies that $(\mathcal{F}, r, k)$ is a "Yes" instance for the Small Union problem. In case 2, there are more than $k + r$ red balls at the root after the moves of g'. Since the number of red balls now exceeds the number of blue balls at the root, each further move of g is of no advantage in decreasing the total score, contradicting that g is a game that achieves a score of at most t.

Theorem 5. Ball and Trap I *remains $W[1]$-hard restricted to binary trees, the maximum number of traps per vertex is one per color, and balls are placed on neither leaves nor parents of leaves.*

The proof follows from straightforward modifications to the construction in Theorem 4. We replace the star T by a binary tree and provide a construction where no internal vertices (with the exception of the root) receive balls or traps.

We introduce the following version of Ball and Trap as it is used to establish the hardness of the Multiple Gene Duplication problem.
Ball and Trap IV (BTIV):
Input: A rooted binary tree T decorated with traps D and balls B in the typical manner, and a positive integer t. *Parameters:* $k = 2$ and the number of traps r. *Conditions:* (1) $|R_b| \leq 2$, f.a. $b \in B$. (2) F.a. $v \in V_T$ and each color c, T_v has at most $|T_v| - 2$ c-colored balls. (3) $R_b = R'_b$ if $l_B(b) = l_B(b')$ and $c_B(b) = c_B(b')$. (4) $R_b \subseteq R_{b'}$ if $l_B(b)$ is an ancestor of $l_B(b')$ in T. (5) No *useless* traps are allowed (a trap d is useless if no ball b in the subtree where the trap is located has $d \in R_b$). (6) If $b, b' \in B$ where the vertex $l_B(b)$ is a descendant of the vertex $l_B(b')$, then all traps $d \in R_b - R_{b'}$ are placed at vertices on the path from b to b' (inclusive). *Question:* Can the Ball and Trap Game be played on T to achieve a score of at most t?

Because none of the conditions in specified in Ball and Trap IV are violated in the reduction, we receive the following corollary.

Corollary 1. Ball and Trap IV *is* $W[1]$*-hard.*

4 Hardness of the Multiple Gene Duplication Problem

Corollary 1 and the following theorem are sufficient to prove $W[1]$-hardness and *NP*-completeness of the Multiple Gene Duplication II problem.

Theorem 6. Ball and TrapIV *reduces to* Multiple Gene Duplication II.

Proof(sketch). We construct an instance $I' \in GDII$ from an instance $I \in BTIV$. Our reduction builds a species tree $S = (V_S, E_S, L)$ and gene trees G_1 and G_2. I is restricted to 2 colors; we associate color 1 with G_1 and color 2 with G_2. W.l.o.g. we restrict our attention to balls and traps of one color c. Let $S = T$. For each vertex $v \in V$, if $v \in L$, then let $free(v) = \{v\}$. When $v \in V - L$ but v is not decorated with a ball or trap, then let $free(v) = free(left(v)) \cup free(right(v))$. If $v \in V - L$ and is decorated by a ball but no trap, then we remove a leaf from $free(left(v))$ and a leaf from $free(right(v))$ and set these to be the children of a new vertex w. For each ball b located at v, we remove an element e from $free(left(v))$ or $free(right(v))$, where e is choosen to be a tree if a tree exists in $free(left(v)) \cup free(right(v))$ or otherwise a leaf. Let w' by the parent of w and e. Let w equal w'. Let $free(w) = free(left(v)) \cup free(right(v)) \cup \{w\}$.

If $v \in V - L$ and is decorated by a trap d, then let e_1 be an element removed from the set $free(left(v))$ and e_2 be an element removed from the set $free(right(v))$. We choose e_1 as follows: If there is a tree in $free(left(v))$ (resp. $free(right(v))$) such that $T_{left(v)}$ ($T_{right(v)}$) has a ball b and $d \in R_b$, then choose one such element. Otherwise, choose any element. It is easy to show that one of e_1 or e_2 must be a tree. Create a new vertex w and place e_1 and e_2 as the children of w. For each ball located at v, we perform the same operations done in the case when $v \in V - L$ and not decorated by a trap. When these operations are completed, we remove one tree from $free(root(T))$; call it τ. It is easy to verify that $free(root(T)) = L - L_\tau$. We complete G_c from τ by embedding the remaining leaves $free(root(T))$ in accordance with the topology of T. We build the maximal subtrees $T_v = (V_{T_v}, E_{T_v}, L_{T_v})$ of T over the elements of $free(\cdot)$. For each such tree T_v we compute the sibling w of v in S and specify p, the lca_τ of the leaves of L_{T_v} in τ. Then we subdivide edge $(p, parent_\tau(p))$ in (p, p') and $(p', parent_\tau(p))$ and add T_v as the sibling of p in τ.

The proof that $I' \in GDII_{``yes''}$ if and only if $I \in BTIV_{``yes''}$ is straightforward and omitted.

5 Conclusions

Via a combinatorial abstraction called the Ball and Trap Game, we have examined the Multiple Gene Duplication problem from both the classical and parameterized complexity frameworks and provided several FPT algorithms for parameterized versions of the problem. It would be interesting to find useful approximation algorithms (or even meaningful heuristics) for parameterized and restricted versions of this problem. Particularly, we would like a nice, meaningful way to guess a (possibly quite large) set of candidate topologies for the species tree S in Multiple Gene Duplication I. One idea is to use the parameterized complexity framework since FPT algorithms are closely related to the design of useful heuristics [4]. Thus far, our models have used unweighted gene and species trees, which means that we have ignored potentially useful *distance data* between genes from the taxa. Future models should include this information. Additionally, the model so far ignores information about the position of genes along the chromosome (it makes no sense to postulate a multiple gene duplication for a set of genes when they lie on different chromosomes in the genome of the organism unless there is evidence that all genes located on this chromosome where duplicated). It may be possible to encode this type (and other types) of restrictions into the Ball and Trap Game. Lastly, other interesting and biologically meaningful parameterizations may lead to good FPT algorithms.

References

1. K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics vol. 147*, pp. 539–564. AMS, 1993. 352
2. S. Benner and A. Ellington. Evolution and Structural Theory. The frontier between chemistry and biochemistry. *Bioorg. Chem. Frontiers* 1 (1990), 1–70. 347
3. R. G. Downey and M. R. Fellows. *Parameterized Complexity*, Springer, 1998. 349, 353
4. R. Downey, M. Fellows, and U. Stege. "Parameterized Complexity: A Framework for Systematically Confronting Parameterized Complexity," in *The Future of Discr. Mathem.: Proc. of the 1st DIMATIA Symp., Czech Republic, June 1997*, Roberts, Kratochvil, Nešetřil (eds.), AMS-DIMACS Proc. Ser. (1998), to appear. 349, 352, 356
5. J. Felsenstein. Phylogenies from Molecular Sequences: Inference and Reliability. *Annu. Rev. Genet.*(1988), 22, 521–65. 347, 348
6. M. Fellows, M. Hallett, C. Korostensky, and U. Stege. "Analogs & Duals of the MAST Problem for Sequences & Trees," ESA 98, to appear. 348
7. W. Fitch, E. Margoliash. "Construction of Phylogenetic Tree," *Sci.* 155 (1967). 347
8. M.Goodman, J.Czelusniak, G.Moore, A.Romero-Herrera, G.Matsuda. "Fitting the Gene Lineage into its Species Lineage: A parsimony strategy illustrated by cladograms constructed from globin sequences," *Syst.Zool.*(1979), 28. 347
9. R. Guigó, I. Muchnik, and T. F. Smith. "Reconstruction of Ancient Molecular Phylogeny," *Molec. Phylogenet. and Evol.* (1996),6:2, 189–213. 347, 348, 349
10. B. Ma, M. Li, and L. Zhang. "On Reconstructing Species Trees from Gene Trees in Term of Duplications and Losses," *Recomb 98*. 347, 348, 349
11. J. Neigel and J. Avise. "Phylogenetic Relationship of mitochondrial DNA under various demographic models of speciation," *Evol. Proc. and Th.*(1986), 515–534. 347

12. R. D. M. Page. “Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas,” *Syst. Biol. 43* (1994), 58–77. 347
13. R. Page, M. Charleston. “From Gene to organismal phylogeny: reconciled trees and the gne tree/species tree problem,” *Molec. Phyl. and Evol. 7* (1997), 231–240. 347
14. L. Zhang. “On a Mirkin-Muchnik-Smith Conjecture for Comparing Molecular Phylogenies,” *J. of Comp. Biol.* (1997) 4:2, 177–187. 347

Visibility Queries in Simple Polygons and Applications[⋆]

Boris Aronov[1], Leonidas J. Guibas[2], Marek Teichmann[3], and Li Zhang[2]

[1] Department of Computer and Information Science, Polytechnic University
Brooklyn, NY 11201-3840, USA
[2] Department of Computer Science, Stanford University, Stanford, CA 94305, USA
[3] Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA

Abstract. In this paper we explore some novel aspects of visibility for stationary and moving points inside a simple polygon P. We provide a mechanism for expressing the visibility polygon from a point as the disjoint union of logarithmically many canonical pieces using a quadratic-space data structure. This allows us to report visibility polygons in time proportional to their size, but without the cubic space overhead of earlier methods. The same canonical decomposition can be used to determine visibility within a frustum, or to compute various attributes of the visibility polygon efficiently. By exploring the connection between visibility polygons and shortest path trees, we obtain a kinetic algorithm that can track the visibility polygon as the viewpoint moves along polygonal paths inside P, at a polylogarithmic cost per combinatorial change in the visibility. The combination of the static and kinetic algorithms leads to a space query-time tradeoff for the visibility from a point problem and an output-sensitive algorithm for the weak visibility from a segment problem.

1 Introduction

In this paper we consider two problems in simple polygons related to visibility — one is the visibility query problem from a fixed observer; the other is the problem of maintaining the visibility from a moving observer. In the following discussion, we denote by n the number of vertices of the simple polygon P and $|V(q)|$ the number of vertices and edges of $V(q)$, the visibility polygon of observer q.

There have been many studies on computing visibility polygons in a simple polygon, and several linear-time algorithms have been proposed. They are optimal in the worst case, as visibility polygons may have up to linear complexity. However, in practice, the visibility polygons are usually much less complex than the environment. In these cases, it is desirable to make the running time of the

[⋆] Boris Aronov has been partially supported by NSF Grant CCR-92-11541 and a Sloan Research Fellowship. Leonidas Guibas and Li Zhang were supported in part by Army Research Office MURI grant 5-23542-A and NSF grant CCR-9623851. Marek Teichmann is supported by the National Science and Engineering Council of Canada.

algorithm output sensitive. That is, we would like the running time of our algorithm to be proportional to the size of the output, after certain preprocessing of the environment.

For the visibility query problem, there are methods for reporting the visibility polygon of a query point in $O(|V(q)| \log n)$ time after $O(n)$ preprocessing time. Note, however, the appearance of a multiplicative overhead in the query bound. In [2] and [7], an optimal query time without multiplicative overhead is achieved. The query time of their algorithms is $O(\log n + |V(q)|)$, but the storage and preprocessing time is cubic in n. In their methods, a full visibility decomposition is built to answer visibility queries. In this paper, we will show how to decompose the polygon into canonical pieces and keep their visibility information separately, so as to reduce the storage and preprocessing time. As a result, our algorithm constructs a data structure of size $O(n^2)$ which can be computed in time $O(n^2 \log n)$ so that the visibility polygon $V(q)$ from any query point $q \in P$ can be reported in $O(\log^2 n + |V(q)|)$ time. In addition, our method for finding the visibility polygon expresses this polygon as the union of $O(\log n)$ canonical pieces, which need not be constructed explicitly unless needed. By exploiting this fact we are able to answer efficiently additional types of queries. For example, we can report the (combinatorial) size of the visibility polygon in $O(\log^2 n)$ time, and obtain output-sensitive visibility queries when visibility is delimited by a cone centered at the observer and defining the viewing frustum.

We also consider the problem of maintaining visibility from a linearly moving viewpoint. A similar problem is studied in [4]. In their problem, the point moves along a given line segment, i.e. the motion is fixed. Here, we allow the motion of the point to be updated in an on-line fashion. By exploiting the intimate connection between visibility and shortest path trees, we obtain an algorithm that can maintain the visibility polygon from a point which moves along linear segments, using linear space and $O(\log^2 n)$ time per event. An event is defined to be either a (combinatorial) change of the visibility polygon, or an update of the linear motion.

Lastly, we give some applications of our methods yielding a storage vs. query-time trade-off for the visibility polygon problem, and an output-sensitive algorithm for computing the weak visibility polygon from a segment.

2 Preliminaries

Suppose P is a simple polygon. Denote by ∂P the boundary of P. For any two points p, q, denote by $\overline{pq}$ the line segment connecting p, q and by $\overrightarrow{pq}$ the ray starting from p and pointing to q. Throughout this paper, we assume non-degeneracy, i.e. no three vertices of P are collinear. Degenerate cases can be handled by standard perturbation techniques.

In the next paragraph, let us assume that p, q are points in P, v is a vertex of P, and s is a line segment inside P. Let n denote the number of sides of P.

The point p is *visible* to q if $\overline{pq}$ lies completely in P. The point p is *weakly visible* to s if p is visible to an *interior* point on s. The *visibility polygon* of p, denoted by $V(p)$, consists of all the points that are visible to p. Likewise, the *weak visibility polygon* of s, denoted by $WV(s)$, consists of all the points that are weakly visible to s. For a visibility or weak visibility polygon V, the *combinatorial representation* of V is a circular list of vertices and edges of P in the order in which they appear on the boundary of V. The *size* of V, denoted by $|V|$, is the number of vertices and edges in the combinatorial representation of V. Two visibility polygons $V(p), V(q)$ are *equivalent* if their combinatorial representations are identical up to a circular permutation. For a vertex v and a point p on ∂P, if $\overline{pv}$ is completely in P, and $q \in \partial P$ is the first point where $\overrightarrow{pv}$ crosses ∂P, the segment $\overline{vq}$ is called the *constraint* induced by p, v. A *critical segment* is a constraint induced by two vertices of P.

We now state some known facts without proof.

Fact 1. *Given the combinatorial representation of $V(p)$ ($WV(s)$), $V(p)$ ($WV(s)$) can be constructed in $O(|V(p)|)$ ($O(|WV(s)| \log n)$) time.*

Fact 2. *All the critical segments partition P into visibility-equivalent cells. This partitioning is called the* visibility decomposition *of P. Further, the combinatorial representations of the visibility polygons of two points in adjacent cells in the visibility decomposition differ in $O(1)$ vertices/edges.*

The critical segments and visibility decomposition play important roles in our paper. Although there might be $\Theta(n^2)$ critical segments, the following crucial fact states that no segment can intersect too many of them.

Fact 3 ([2,7]). *Any segment s inside a simple polygon P can cross at most $O(n)$ critical segments of P.*

Another geometric object which is closely connected to visibility is the shortest path. For a simple polygon P and two points $p, q \in P$, the *shortest path* $\pi(p, q)$ is the path with the shortest Euclidean length among all the paths in P that join p, q. It is well known that $\pi(p, q)$ is a polygonal path in which all the intermediate vertices are reflex polygon vertices. The tree formed by the union of $\pi(p, v)$, for all the vertices v on ∂P, is called the *shortest-path tree* rooted at p and denoted by $T(p)$. For two vertices u, v, u is v's parent in $T(p)$ if u is the immediate predecessor of v on $\pi(p, v)$. An immediate connection between $V(p)$ and $T(p)$ is that the vertices of P visible from p are exactly the children of p in $T(p)$. In Section 4, we will see more connections between $T(p)$ and $V(p)$.

In the paper, we use persistent data structures [10] to reduce storage requirements. A persistent data structure is one that accepts an arbitrarily long sequence of updates, but is able to remember at any time all its earlier versions. Here, we focus our attention on persistent red-black trees. Suppose we have a set of m linearly ordered items and a sequence of ℓ updates (i.e. insertions and deletions) on these items. Let the version at time t, for $1 \leq t \leq \ell$, be the set resulting from applying the first t updates in the sequence to an empty set. The following fact is due to Sarnak and Tarjan [10].

Fact 4 ([10]). *A persistent red-black tree can be built so that any version can be accessed with the same time bounds as if the items were stored in a normal red-black tree. Furthermore, the structure can be constructed in $O((m+\ell)\log m)$ time, using $O(m+\ell)$ space.*

3 Answering Visibility Queries

In what follows, we provide an algorithm with $O(\log^2 n + |V(q)|)$ query time, which is nearly optimal and requires $O(n^2 \log n)$ preprocessing time and $O(n^2)$ storage. Compared to the algorithms in [2,7], the storage and preprocessing time has one fewer linear factor. Further, we show that the algorithm can be extended to handle the problems of counting the size of a visibility polygon and answering *cone-visibility* queries.

Intuitively, the visibility polygon of a query point is computed by first partitioning the interior of the polygon into canonical regions, then constructing the visibility polygon in each region, and finally gluing them together. In Section 3.1, we show how to compute the visibility polygon inside a region. In Section 3.2, we describe the balanced triangulation which serves as the decomposition and present the full algorithm and its extension to cone-visibility queries.

3.1 Computing Partial Visibility Polygons

For a polygon $Q \subset P$, define the *partial visibility polygon* $V_Q(q)$ to be the polygon $V(q) \cap Q$. Suppose P is divided into two parts, L and R, by a diagonal e. In this section, we will show how to compute the partial visibility polygon $V_L(q)$ of a point $q \in R$.

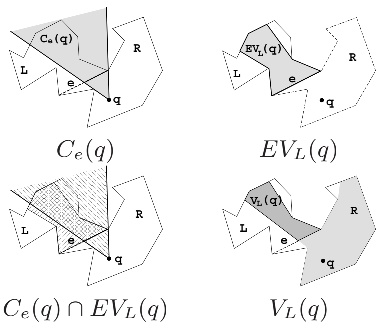


Fig. 1. Computing the partial visibility polygon

Observe that for any two points $p \in L$ and $q \in R$, q can see p if and only if the line segment $\overline{pq}$ doesn't cross ∂P. Since $\partial P = (\partial L \setminus \{e\}) \cup (\partial R \setminus \{e\})$, we consider them separately. Define the *visibility cone* $C_e(q)$ to be the cone with apex q and delimited by the endpoints of the portion of e visible from q. Then, $\overline{pq}$ does not cross $\partial R \setminus \{e\}$ if and only if p lies inside the cone $C_e(q)$ because e separates R from L. On the other hand, if we define the *exterior visibility polygon* $EV_L(q)$ of q with respect to L to be the portion of L which can be seen by q as if all the edges of ∂R are transparent, then $\overline{pq}$ doesn't cross $\partial L \setminus \{e\}$ if and only if $p \in EV_L(q)$. Therefore, Figure 1.

Computing $C_e(q)$. Because of the simplicity of P, the portion of e visible from q is a line segment delimited by the extensions of the edges incident to q on the shortest paths from q to the endpoints of e. If we take possible collinearity into account, it suffices to consider the two edges nearest to q on each shortest path. We omit the details in this version of the paper. In [5], it is shown that a data structure can be built in linear time and using linear space so that, for any two query points, the length of the shortest path between them can be reported in $O(\log n)$ time. Using the same structure, we can obtain the first two edges on the shortest path. Thus, $C_e(q)$ can be computed in $O(\log n)$ time after $O(n)$ time preprocessing.

Computing $\boldsymbol{EV_L(q)}$. By analogy with the visibility decomposition, we can decompose the exterior of L into cells so that two points in the same cell see equivalent visibility polygons in L. This decomposition is called *exterior visibility decomposition* and denoted by $\mathcal{EV}_L$. Once we have constructed $\mathcal{EV}_L$, $EV_L(q)$ can be computed by locating q in the cell decomposition and retrieving the corresponding exterior visibility polygon.

Consider all the critical segments induced by pairs of vertices in L. We only need to consider those segments that cut e. We extend all those critical segments into the open space that does not contain L and obtain half lines, which are called *critical rays*. The arrangement of all these rays gives us the exterior visibility decomposition similar to that in Fact 2. According to Fact 3, the number of such critical rays is $O(n)$. This key fact reduces the complexity of the decomposition to $O(n^2)$, which can be computed in $O(n^2)$ time.

For each cell in $\mathcal{EV}_L$, we compute and store the corresponding visibility polygon. If we do it in a naive way, it may take $\Theta(n^3)$ space and preprocessing time because a visibility polygon may have $\Theta(n)$ complexity. In the following, we will show how to reduce the costs by using persistent data structures. We form a tour Γ visiting all the cells of $\mathcal{EV}_L$ so that each cell appears on Γ at most twice. Since the visibility polygons of points in two adjacent cells on Γ have only $O(1)$ differences, the visibility polygon of one cell can be obtained from the other by $O(1)$ updates. Thus, we can start from an arbitrary cell, walk along Γ, and construct a persistent red-black tree on the combinatorial representation of the visibility polygons for all the cells. The structure takes $O(n^2)$ storage and can be built in $O(n^2 \log n)$ preprocessing time by Fact 4. In addition, we build a point location structure on top of the arrangement.

To answer a query q, we first locate the cell of $\mathcal{EV}_L$ containing q, then retrieve the corresponding root pointer into the persistent data structure. Both steps can be done in $O(\log n)$ time. Therefore, we can preprocess an n-gon L and an edge e of L into a data structure by using $O(n^2)$ space and $O(n^2 \log n)$ preprocessing time so that for any query point $q \notin L$, a pointer pointing to a red-black tree which stores $EV_L(q)$ can be returned in $O(\log n)$ time.

Computing $\boldsymbol{V_L(q)}$. Once we have computed $EV_L(q)$ and $C_e(q)$, $V_L(q)$ can be computed by extracting the portion of $EV_L(q)$ inside the cone $C_e(q)$. Because the visibility polygon is star-shaped, and the visible vertices and edges are stored in a red-black tree in the order of their appearance on the visibility polygon, the pruning procedure amounts to reporting all the elements of a red-black tree within a given range of keys. For the persistent data structure in [10], this can be done, for any version, in $O(\log n + k)$ time where k is the output size. Therefore, we have

Theorem 1. *Given a polygon P and a diagonal e which cuts P into two parts, L and R, by using $O(n^2 \log n)$ time, we can construct a data structure of size $O(n^2)$ so that, for any query point $q \in R$, the partial visibility polygon $V_L(q)$ can be reported in $O(\log n + |V_L(q)|)$ time.*

3.2 Computing Visibility Polygons by Balanced Triangulation

In Section 3.1, we showed how to compute a partial visibility polygon. In this section, we show how to combine it with a balanced triangulation to compute the entire visibility polygon.

The *balanced triangulation* of a simple polygon P is based on the observation that there always exists a diagonal e of a simple polygon P that cuts P into two pieces, each with at most $2n/3$ vertices [3]. By recursively subdividing, a balanced binary tree can be created where all the leaves are triangles and each interior node i corresponds to a sub-polygon P_i and a diagonal e_i. The left and right subtrees of i correspond to two sub-polygons, L_i, R_i, obtained by cutting P_i along e_i. For each interior node i in the tree, we build the data structures as described in Section 3.1 for reporting partial visibility polygon in L_i and R_i with respect to the diagonal e_i. For simplicity, denote by $V_i(q)$ the partial visibility polygon $V_{P_i}(q)$.

To compute $V(q)$, we first locate q among the leaf triangles. Let the path, from the leaf to the root, be $i_1(\text{leaf}), i_2, \cdots, i_k(\text{root})$. We construct all the $V_i(q)$'s for i in the path inductively. For the leaf node, it is simply the corresponding triangle. In the inductive step, suppose that we have constructed $V_{i_j}(q)$, and i'_j is i_{j+1}'s other child. Since q is not in $P_{i'_j}$, we can query the partial visibility query data structure stored in i_{j+1} to obtain $V_{i'_j}(q)$. Then, by gluing $V_{i_j}(q)$ and $V_{i'_j}(q)$ together, we can obtain $V_{i_{j+1}}(q)$. The gluing can be done in $O(\log n)$ time by representing each $V_i(q)$ as a circular list and storing in an auxiliary array the pointers pointing to the diagonal edges appearing on $V_i(q)$.

Thus, we have

Theorem 2. *A simple polygon P can be processed into a data structure using $O(n^2)$ space and within $O(n^2 \log n)$ time so that for any query point q, $V(q)$ can be reported in time $O(\log^2 n + |V(q)|)$, and the size of $V(q)$ can be reported in $O(\log^2 n)$ time.*

Proof. If we denote by $S(n), T(n)$ the space and time needed for building the data structure mentioned above for an n-gon, respectively, then we have the following recurrences

$$S(n) = S(m) + S(n-m) + \Theta(n^2)\,, \quad T(n) = T(m) + T(n-m) + \Theta(n^2 \log n)\,,$$

where $n/3 \le m \le 2n/3$. Therefore, $S(n) = \Theta(n^2)$, and $T(n) = \Theta(n^2 \log n)$.

As for the query time, the point location can be performed in $O(\log n)$ time. In addition, because the triangulation is balanced, the length of any path from the root to a leaf is $O(\log n)$. For each node i, the time needed to query the structure $V_i(q)$ is $O(\log n + |V_i(q)|)$ as shown in Theorem 1. Each merging can be done in $O(\log n)$ time according to the above discussion. Therefore, in total, the query time is $O(\log n + \sum_i (\log n + |V_i(q)|)) = O(\log^2 n + |V(q)|)$. Obviously, $|V(q)|$ can be reported in $O(\log^2 n)$ time. □

As an application, the above method can be extended to the *cone visibility query problem*, where, in addition to a query point q, a query also includes a cone with q as the apex which delimits the visibility of q. We are asked to report the visibility from q within the cone. This can be done in the same space and time bound as above — to answer a cone visibility query, still we compute partial visibility polygons and glue them together. The only difference is that we need to prune the exterior visibility polygon by both $C(q)$ and $C_e(q)$ to get the partial visibility polygon. We can first overlay two cones $C(q), C_e(q)$ to obtain a single cone and use it to perform a range search.

Corollary 1. *Given a simple polygon P, we can process it into a data structure with $O(n^2)$ space and in $O(n^2 \log n)$ time so that for any query point q and a cone $C(q)$, the visible region from q within the cone $C(q)$ can be reported in time $O(\log^2 n + k)$ where k is the output size.*

4 Maintaining Visibility from a Moving Viewpoint

In this section, we present an algorithm to maintain the visibility polygon from a moving viewpoint. Suppose that we have a point p inside the polygon P, and p moves along a line segment. We will describe a data structure by which the visibility can be maintained correctly as time goes on. In addition, our algorithm maintains the visibility in an on-line fashion, namely, once the motion of the point changes, the data structure can be updated efficiently. In fact, our algorithm fits the framework of kinetic data structures in [1] very well and satisfies all the efficiency criteria proposed in their paper.

4.1 Combinatorial Changes of the Shortest-Path Tree

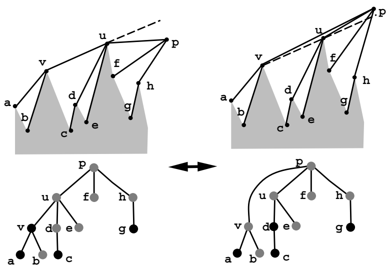


Fig. 2. Events in the maintenance of the shortest-path tree

Instead of maintaining the visibility polygon, we will maintain the shortest-path tree of p. Recall that the shortest-path tree $T(p)$ is the tree rooted at p formed by taking the union of the shortest path from p to every vertex of P. We define the *principal child* vertex (edge) of a non-root node v of $T(p)$ to be the vertex w (resp. edge $\overline{vw}$) of $T(p)$ among the children of v such that the angle formed by $\overrightarrow{vw}$ and $\overrightarrow{uv}$, where u is the parent of v, is the smallest among all such angles. For example, in Figure 2, all the principal children are darkened in the tree.

On the combinatorial changes of the shortest-path tree and their connection to the changes of the visibility polygon, we have the following result.

Lemma 1. *As p moves in P, $V(p)$ changes iff $T(p)$ changes combinatorially iff the set of first-level vertices of $T(p)$ changes. This occurs exactly when either a pair of consecutive first-level vertices of $T(p)$ become collinear with p, or a first-level vertex and its principal child become collinear with p.*

The proof of Lemma 1 is omitted in this abstract. For an illustration, refer to Figure 2 for what happens before and after p is collinear with u, v.

4.2 Tracking Visibility Changes

According to Lemma 1, it is sufficient to check when the point p is collinear with $O(|V(p)|)$ pairs of vertices to detect when $T(p)$ (and thus $V(p)$) changes. The checking of collinearity is equivalent to detecting when p crosses the lines defined by those pairs. Let us call those lines *constraint lines.* Consider the convex face τ in the arrangement formed by all the constraint lines which contains the current p. To cross a constraint line, p has to cross an edge on the boundary of τ. Thus, once we can maintain τ when p moves, for any change of p's motion, a binary search on the boundary of τ will tell us which edge is the next one p is going to cross, provided p keeps moving along the current direction. By duality, τ can be maintained by a dynamic convex hull algorithm with $O(\log^2 n)$ cost per update. Therefore, we obtain the following algorithm.

Preprocessing. First, we construct the shortest-path tree $T(p)$ from the initial position of p to all vertices of P as in [6]. Further, obtain, for each vertex v, the doubly-linked list of its children, sorted around v, with pointers to the first and last. In each vertex, we store a pointer to its principal child.

We also compute $V(p)$ of the initial position of p and store the combinatorial representation of $V(p)$ in a red-black tree according to the order they appear on ∂P. Clearly, all the above processing can be done in $O(n)$ time.

By examining the first-level vertices of the shortest-path tree $T(p)$ and their principal children, we collect a set of $O(n)$ lines as per Lemma 1 and compute the convex face τ in the arrangement that contains p. We store (the dual of) τ into a dynamic convex hull structure of [9], which can be initialized in $O(n \log n)$ time using $O(n)$ space and updated in $O(\log^2 n)$ time per insertion and deletion. We also compute the time when p crosses the boundary of τ along the current direction.

Updating. There are two types of events that may happen.

The point p crosses a constraint. As per Lemma 1, this is when $T(p)$ changes. According to which constraint p crosses, we update $T(p)$ as shown in Figure 2. Namely, we either cut a principal child v from a first-level vertex u and insert v right before or after u, as appropriate, as a first-level vertex or perform the reverse operation, depending on the direction that p crosses the constraint. These updates can be done in $O(1)$ time. Furthermore, the structures are updated by eliminating and adding the appropriate constraints into the dynamic data structure representing the face τ that contains p. Since there are only $O(1)$ changes on $T(p)$, this operation takes $O(\log^2 n)$ time [9]. Finally, we compute the next time when p crosses the boundary of τ by performing a binary search on the updated τ in $O(\log n)$ time. To maintain $V(p)$, we need to maintain the edges visible to p as well as the vertices. Note that once p crosses a constraint line that passes through the vertices u, v, only the visibility of the edge hit by the extension of $\overrightarrow{uv}$(or $\overrightarrow{vu}$) can change. Therefore, a ray-shooting data structure, such as the one in [8], suffices to detect which edge it is in $O(\log n)$ time.

The motion of p is updated. Since p's motion is linear, we perform a binary search to determine which edge on the boundary of τ p is going to cross next. This can be done in $O(\log n)$ time.

To summarize, the data structure can be updated in total $O(\log^2 n)$ time for each actual change in the shortest-path tree and the view and for each change of the motion. In fact, in the above bounds $\log n$ can be replaced by $\log k$, where k the number of currently relevant constraints, which is proportional to the size of $V(p)$, for the current position of point p. Thus we have

Theorem 3. *Let P be a simple n-gon and p an initial position in P. After $O(n \log n)$ time and $O(n)$ space preprocessing, for a point undergoing linear motion, the time when the first combinatorial change in $V(p)$ (or, equivalently, in $T(p)$) can be determined in $O(\log |V(p)|)$ time. The entire data structure can be updated in $O(\log^2 |V(p)|)$ time per change.*

5 Applications

In the previous sections we provided algorithms for answering visibility queries and maintaining the visibility polygon from a moving viewpoint. In this section, we give some applications of the combination of these methods. Due to space constraints, we will only state our results and leave the details for the full version of this paper.

As for the visibility query problem, instead of computing each exterior visibility decomposition, we can compute and store a coarser arrangement (cutting), an $(1/r)$-*cutting* of the exterior visibility decompositions. Then, for any given query point q, instead of locating it in the exterior visibility decomposition, we retrieve the visibility polygon of a nearby point and walk from that point to q to construct $V(q)$. We can obtain a time-space tradeoff as follows.

Theorem 4. *Given a simple n-gon and m between $n \log^3 n$ and $n^2 \log n$, one can preprocess the polygon in $O(m \log n)$ time and $O(m)$ space so that, for a query point q, $V(q)$ can be computed in $O((n^2/m) \log^3 n + |V(q)|)$ time.*

As another application, we consider the *weak visibility queries* of line segments. For a query segment s, we can walk on s, from one endpoint to the other, to compute its weak visibility in an output-sensitive time by gluing together all the visibility polygons of the points on s. The result is:

Theorem 5. *Given a simple n-gon P in the plane, one can preprocess P in $O(n^2)$ space and $O(n^2 \log n)$ time so that, for any segment s inside P, $WV(s)$ can be computed in $O(|WV(s)| \log^2 n)$ time.*

6 Conclusion

In the paper, we showed how to answer visibility queries in nearly optimal time by using quadratic preprocessing time and storage, which improves by a linear factor on the previous algorithms. Then, we provided an algorithm which can maintain visibility polygon from a moving point in linear space and $O(\log^2 n)$ time per change in visibility or flight plan. A nice property of our method of answering visibility queries is that the visibility polygons are represented by a small number of canonical parts, each is stored in a form suitable for searching. By exploiting this fact, we obtained algorithms for some other visibility problems. As applications of both methods, we can obtain a space query-time tradeoff for visibility query and an output-sensitive algorithm for the weak visibility from a segment query problem.

An immediate open question is whether we can improve the space and preprocessing time further to a nearly linear bound while keeping the query time the same, i.e. a poly-logarithmic additive overhead and $O(1)$ cost for each output. As to the maintenance problem, our bound relies on the fact that the motion is linear. The situation when the motion is a general algebraic curve remains open. For both problems, it is interesting to know if our techniques can be extended to the case of polygons with holes.

References

1. J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pages 747–756, 1997. 363
2. P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 23–28, 1992. 358, 359, 360
3. B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339–349, 1982. 362
4. D. Z. Chen and O. Daescu. Maintaining visibility of a polygon with a moving point of view. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 240–245, 1996. 358
5. L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39:126–152, 1989. 361
6. L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. 364
7. L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem in two dimensions. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 259–268, 1992. 358, 359, 360
8. J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 54–63, 1993. 365
9. M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983. 365
10. N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29:669–679, 1986. 359, 360, 362

Quadtree Decomposition, Steiner Triangulation, and Ray Shooting[⋆]

Siu-Wing Cheng and Kam-Hing Lee

Department of Computer Science, HKUST
Clear Water Bay, Hong Kong

Abstract. We present a new quadtree-based decomposition of a polygon possibly with holes. For a polygon of n vertices, a truncated decomposition can be computed in $O(n \log n)$ time which yields a Steiner triangulation of the interior of the polygon that has $O(n \log n)$ size and approximates the minimum weight Steiner triangulation (MWST) to within a constant factor. An approximate MWST is good for ray shooting in the average case as defined by Aronov and Fortune. The untruncated decomposition also yields an approximate MWST. Moreover, we show that this triangulation supports query-sensitive ray shooting as defined by Mitchell, Mount, and Suri. Hence, there exists a Steiner triangulation that is simultaneously good for ray shooting in the query-sensitive sense and in the average case.

1 Introduction

Triangulation is a popular research topic because many problems call for a decomposition of a scene into simple elements that facilitate processing. In the plane, the focus has been on optimizing or approximating some measure. Steiner points are allowed and the triangulation obtained must respect the given line segments (i.e., a given line segment must be equal to the union of some edges in the triangulation). One natural measure is the total length of edges in the triangulation. This is called the *weight* of the triangulation. No algorithm is known that computes the *minimum weight Steiner triangulation* (MWST for short) of a point set or a polygon. Eppstein [5] presented an $O(n \log n)$-time algorithm to approximate the MWST of a point set to within a constant factor. A variant of the method also works for a convex polygon. One question raised in [5] is how to compute an approximate MWST for a general polygon.

The MWST problem, which has been of theoretical interest so far, is recently related to the *ray shooting* problem by Aronov and Fortune [1]. The ray shooting problem is to report the first obstacle hit by a query ray. In this paper, we assume that the scene is a polygon possibly with holes. Query light source will fall into this polygon, and the boundary of the polygon acts as obstacles. The usual scene of a collection of simple polygons can be modeled by enclosing them in a square and treating the given polygons as holes.

⋆ Research supported in part by RGC CERG HKUST 650/95E.

A simple approach for the ray shooting problem is to decompose the polygon into a *planar subdivision* so that each face is of constant complexity and has at most a constant number of adjacent faces. Then the ray shooting query is answered by walking from face to face until the first obstacle is hit. Aronov and Fortune [1] showed that a Steiner triangulation of small weight is good for ray shooting in the average case. The average is taken over all random choices of light source on polygon boundary and over all random choices of ray direction. In 2D, the average number of triangulation edges visited is equal to the weight of Steiner triangulation divided by the boundary length of the polygon. Aronov and Fortune also claimed a polynomial-time algorithm for approximating the MWSTs of a set of line segments enclosed within their convex hull in 2D.

Mitchell, Mount, and Suri [7] proposed a notion of *C-complexity* for measuring the complexities of the scene and a ray shooting query. In the plane, a *C-ball* is a connected component of the intersection the scene with a disk. The center and radius of the C-ball are taken to be the center and radius of the defining disk. Given a C-ball B, if we expand its defining disk by a factor of $1+\epsilon$, then there is one connected component in this expanded disk that contains B. We denote this component by $(1+\epsilon)B$. A C-ball is *simple* if it does not intersect more than two edges. A C-ball B is ϵ-strongly simple if both B and $(1+\epsilon)B$ are simple. For a fixed ϵ, the C-complexity of a polygon is the minimum number of ϵ-strongly simple C-balls that cover the interior of the polygon. Mitchell *et al* presented an algorithm to decompose the polygon interior into convex cells so that given a query ray, the number of cells visited is proportional to the number of ϵ-strongly simple C-balls that cover the ray up to the first intersection. This is called *query-sensitive ray shooting* in [7].

The main contribution of our paper is a new quadtree-based decomposition of a polygon possibly with holes which is inspired by the quadtree decomposition developed for multiple-tool milling [2]. We prove that triangulating the decomposition yields an approximate MWST of the polygon. For a polygon of n vertices, a truncated decomposition has $O(n \log n)$ size and can be computed in $O(n \log n)$ time. Triangulating the truncated decomposition also produces an approximate MWST of size $O(n \log n)$. The boundary length of the polygon is included in the weight of the triangulation. No result is known that excludes the boundary length. Our result is different from that in [1] since they require a convex outer boundary and they compatibly triangulate the holes too. In general, compatible triangulation of the holes produces more triangles (possibly $\Omega(\log n)$ more, see Section 5).[1] We also show that the triangulation of the untruncated decomposition supports query-sensitive ray shooting. Since this triangulation also approximates the MWST, this demonstrates the existence of a Steiner triangulation that is simultaneously good for ray shooting in the query-sensitive sense and in the average case.

The rest of the paper is organized as follows. Section 2 provides the basic definitions. Section 3 describes our quadtree-based decomposition. In Section 4, we

[1] In other words, we assume that the exterior of the polygon is inaccessible by light. This is not assumed in [1] as each line segment is treated as an individual obstacle.

discuss how to use the decomposition to obtain approximate MWSTs. Section 5 discusses the application in ray shooting.

2 Preliminaries

Let P denote a polygon possibly with holes. Our quadtree-based decomposition can accommodate degenerate holes. However, if a Steiner triangulation is to be obtained, then we allow a hole degenerate to a single point but not line segments.

A *quadtree* is a ternary tree representing a hierarchical decomposition of the plane, originally proposed for representing point sets. Each node of the quadtree corresponds to a square region, called a *box*. The root usually corresponds to the smallest enclosing square of the given set of objects. In our case, it is the smallest enclosing square of P. A node of the quadtree acquires four children when its associated box is split into its four quadrants. Given a box b, we denote its width by $\mathit{size}(b)$ and its parent by $p(b)$.

In the quadtree, boxes of the same size are of the same height and they form a regular grid partitioning the entire plane. A *neighbor* of a box b is a box of the same size that touches b. An *orthogonal neighbor* of a box b is a box of the same size that shares a side with b.

For any box b and a constant c, we denote by $c \cdot b$ a square with the same center as b and side length $c \cdot \mathit{size}(b)$. Given any square S (not necessarily a box in a quadtree), we call a connected component of $P \cap S$ a *subpolygon* of S. We define the size of a subpolygon α of a box b to be $\mathit{size}(b)$ and denote it by $\mathit{size}(\alpha)$.

3 Quadtree-Based Decomposition

For each subpolygon α of b, define $\mathit{zone}(\alpha)$ to be the subpolygon of $3b$ that contains α. The decomposition takes a parameter $\delta > 0$ which is fixed beforehand. A subpolygon α of b is *crowded* if $\mathit{size}(\alpha) > \delta$ and $\mathit{zone}(\alpha)$ contains more than one polygon vertex. If α is uncrowded and the subpolygon of $p(b)$ containing α is crowded, then α is a *cell* in the final decomposition. We say the the cell α is *created* at box b and we call b the *home box* of α.

A box b is crowded if one of its subpolygons is crowded. A crowded box is split into its four quadrants, thus generating four children in the quadtree. *Only* the crowded subpolygons of b will be split together with b. The components obtained will be distributed into the four children of b. Although a cell created at b will not be split further, new vertices may be inserted into its boundary when adjacent crowded subpolygons are split. When no box in the tree is to be split further, the collection of cells created form a planar subdivision of the interior of P. We denote this by $\mathcal{D}$. Figure 1 illustrates one step of this hierarchical decomposition.

If we set the parameter δ to be sufficiently small (e.g. less than the smallest Euclidean distance between two polygon vertices), then each cell has at most one polygon vertex. In this case, we call $\mathcal{D}$ *untruncated*. Otherwise, $\mathcal{D}$ is *truncated*. In $\mathcal{D}$, a cell is *normal* if its size is larger than δ, otherwise it is *small*. Normal

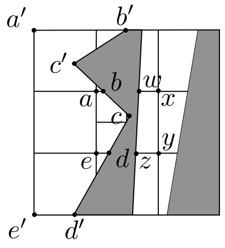


Fig. 1. The shaded regions belong to holes. The subpolygon $wxyz$ is not crowded and so it becomes a cell. The subpolygon $abcde$ is crowded as its zone $a'b'c'cd'e'$ contains two polygon vertices. So $abcde$ is split further when splitting the box. Note that $wxyz$ is not split.

cells have constant complexity and small cells have the same size. There are two kinds of edges bounding a cell. Those that lie on some polygon edges are called *solid edges* and the others are called *non-solid edges*.

Our hierarchical decomposition proceeds in one pass in a top-down fashion. There are two key steps, namely crowdiness testing and subpolygon splitting. To facilitate the subsequent splitting of subpolygons, we compute a noncrossing spanning tree of the holes and the outer boundary of P. This is done as follows. First, we convert P to a monotone subdivision by adding diagonals in $O(n \log n)$ time [8]. Then we invoke a linear time graph search of the subdivision to identify a spanning tree of the holes and the outer boundary of P. Splitting along the spanning tree edges yields a degenerate simple polygon P^* of size $O(n)$. P^* will be useful later.

We split all crowded subpolygons of boxes of the same size before continuing to the next level. Inductively, a list of subpolygons of a box b will be inherited from the splitting of crowded subpolygons of $p(b)$. Take a subpolygon α in the list. If α contains more than one polygon vertex, then α is crowded. Suppose not. Then we collect subpolygons that touch α. There are two kinds. The first kind has size larger than $size(\alpha)$. These are normal cells of constant complexity. There are at most three such cells touching α, and it can be checked in constant time whether they contribute more than one polygon vertex in $zone(\alpha)$. The second kind are subpolygons of neighbors of b that touch α. These subpolygons are subsets of $zone(\alpha)$. For each such subpolygon, its number of polygon vertices can be precomputed. So it takes constant time to check each such subpolygon. Moreover, since α is assumed to contain at most one polygon vertex (α is already crowded otherwise), there are at most nine subpolygons of neighbors of b that touch α. In all, testing crowdiness of α takes constant time. If α is uncrowded, then it is a cell and we remove it from the list. Otherwise, α will be split.

Splitting α is equivalent to clipping α within each of the four quadrants of b. We discuss how to do this for one quadrant. If α is a simple polygon, then this can be done in linear time using Jordan sorting [6]. However, α may contain holes and in this case, P^* comes to our rescue. We keep an *unnested version* for α defined as follows. The spanning tree edges in P^* separate α into several components. Take any two such components that are separated by a spanning tree edge that cuts completely across b. Remove this separating tree edge and merge the two components into one. Continue until there is no such adjacent pair of components. (See Figure 2.) The resulting set of simple polygons is the unnested version of α and we denote it by α^*. Note that α and α^* covers the same region in the plane.

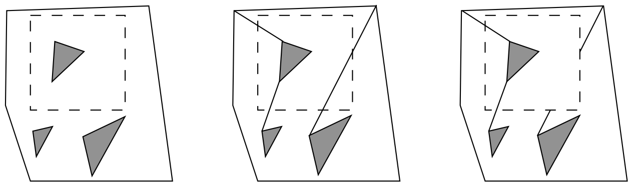


Fig. 2. In the left figure, the shaded regions are holes in the polygon. The dashed square is a box which has only one subpolygon α. After adding the spanning tree edges in P^*, we obtain the middle figure. α is separated into three components and two are separated by a spanning tree edge that cuts completely across the box. In the right figure, after merging, we obtain two simple polygons which form α^*.

We clip each simple polygon in α^* instead of α. This can be done using Jordan sorting [6]. Let m be the size of α. The total complexity of α^* is $O(m)$ as any spanning tree edge bounding α^* is incident to a vertex of α. This implies that the clipping takes $O(m)$ time. What remains is how to pierce together the components resulting from clipping α^* to produce the desired subpolygons and their unnested versions. As before, we repeatedly merge any two components that are separated by a spanning tree edge that cuts completely across the quadrant. This takes time linear in the number of such spanning tree edges which is $O(m)$ as they all bound α^*. Let X be the set of simple polygons produced. Identify all the maximal connected subsets of polygons in X in linear time. The union of each maximal connected subset is a subpolygon of the quadrant whose unnested version is the maximal connected subset. In all, clipping α can be done in linear time.

$\mathcal{D}$ satisfies a structural property: no cell can be bounded by a collinear chain of three non-solid edges. This implies that every normal cell and every small cell containing at most one polygon vertex is adjacent to a constant number of other cells. This is similar in spirit to the “balance” or “smoothness” properties of other quadtree decompositions studied [3,5,7].

Lemma 1. *No cell can be bounded by a collinear chain of three non-solid edges.*

Proof: Assume to the contrary there is a collinear chain s of three non-solid edges bounding a cell γ. Let b_1 be the home box of γ. Since s contains quadtree vertices, b_1 does not have the smallest size in the hierarchy and so γ is a normal cell. Let b_2 be the orthogonal neighbor of b_1 that shares s. There must be some proper descendant b_3 of b_2 such that b_3 puts a corner vertex on s, and a proper descendant of b_3 puts another corner vertex v on s. Let α be the subpolygon of b_3 that contains the point v on its boundary. Since α was split, $\mathit{zone}(\alpha)$ contains more than one polygon vertex. Since α is adjacent to γ and $\mathit{zone}(\alpha) \subseteq 3b_3 \subseteq 3b_1$, we conclude that $\mathit{zone}(\gamma)$ also contains more than one polygon vertex, contradiction. $\square$

4 Analysis

4.1 Total Length of the Decomposition

We conceptually refine $\mathcal{D}$ to another decomposition $\mathcal{D}^*$ to ease the analysis. This is done by relaxing the definition of crowdiness. A subpolygon α is crowded if $\mathit{size}(\alpha) > \delta^*$ and $\mathit{zone}(\alpha)$ contains *at least one* polygon vertex (instead of more than one), where δ^* is a constant chosen to be less than δ and close to zero. At the end, we also call a cell γ of $\mathcal{D}^*$ normal if $\mathit{size}(\gamma) > \delta^*$ and small otherwise. Since $\mathcal{D}^*$ is a refinement of $\mathcal{D}$, its total length is larger and so it suffices to bound the total length of $\mathcal{D}^*$. The advantage of $\mathcal{D}^*$ is that any cell that has a polygon vertex must be small.[2] We first state a technical lemma. Its proof is omitted here.

Lemma 2. *Let γ_1 be a normal non-square cell of $\mathcal{D}^*$. Suppose that each solid edge of γ_1 has length less than $\mathit{size}(\gamma_1)/4$. Then γ_1 is adjacent to a cell γ_2 of $\mathcal{D}^*$ of perimeter $\Theta(\mathit{size}(\gamma_1))$ and the total length of solid edges bounding γ_2 is at least $\mathit{size}(\gamma_1)/4$.*

We can now bound the total length of $\mathcal{D}^*$ and hence the total length of $\mathcal{D}$. Our proof follows the approach in [5].

Lemma 3. *For a polygon P of n vertices, the total length of $\mathcal{D}^*$ and hence $\mathcal{D}$ is bounded by $O(w(MWT))$, where $w(MWT)$ is the weight of the minimum weight triangulation of P.*

Proof: Let MWT denote the minimum weight triangulation of P. Recall that no Steiner point is allowed in MWT. Small cells have negligible perimeter as δ^* is close to zero. So we focus on normal cells.

Let γ be a normal square cell. Let Δ be the triangle in MWT that covers the center of γ. If Δ has a vertex u inside 11γ, then we charge the perimeter of γ proportionally to the two edges of Δ incident to u. Since the vertices of Δ are outside γ, at least one edge of Δ has length $\mathit{size}(\gamma)$. Thus, the total length of the two edges incident to u is large enough to absorb the charge. Suppose that Δ does not have a vertex inside 11γ. Let α be the subpolygon of $p(\gamma)$ that contains γ. Since α was split, $\mathit{zone}(\alpha)$ contains a polygon vertex which implies that $\mathit{zone}(\alpha)$ has a polygon vertex v that can see the center, c, of γ. The line

[2] The disadvantage is that the excessive splitting makes the size of $\mathcal{D}^*$ not bounded by the C-complexity of P.

segment cv stabs a sequence of triangles in MWT. We visit this sequence in order starting from Δ. Eventually, we must reach a triangle Δ' with a vertex inside 11γ as cv has length less than $5.25 size(\gamma)$. Let xy be the edge of Δ' through which we step into Δ'. So x and y are outside 11γ. Since x and y are at distance at least $5.5 size(\gamma)$ from c, xy has length at least $\sqrt{5.5^2 - 5.25^2} size(\gamma) > size(\gamma)$. Thus, we can charge the perimeter of γ proportionally to the two edges of Δ' incident to the vertex opposite to xy.

In the above charging scheme, if an edge e of MWT acquires a charge O(length of e) from a square cell γ, then γ lies inside a box of width $12 size(\gamma)$ centered at an endpoint of e. At one level of the quadtree, there are at most $2 \times 12^2 = 288$ square cells that satisfy this requirement. Since box size halves from level to level, the accumulated charge at e telescopes to O(length of e). This shows that the sum of perimeter of normal square cells is $O(w(MWT))$.

Consider a non-square cell γ_1. If the solid edges bounding γ_1 have a total length at least $size(\gamma_1)/4$, then we charge the perimeter of γ_1, which is $O(size(\gamma_1))$, to these solid edges proportionally. When this is not the case, we apply Lemma 2 and charge the perimeter of γ_1 to the perimeter of an adjacent cell.

In all, the total length of $\mathcal{D}^*$ is asymptotically bounded by the sum of $w(MWT)$ and the boundary length of P. Since $w(MWT)$ includes the boundary length of P, the result follows. □

4.2 Approximate MWST

To obtain an approximate MWST, we triangulate the cells in $\mathcal{D}$. By Lemma 1, triangulating a cell with at most one polygon vertex adds only a constant number of diagonals. Thus, the total length will be increased by only a constant factor. Triangulating a cell with $m > 1$ polygon vertices may add $O(m)$ diagonals. However, each such diagonal has length at most δ. Summing over all cells with more than one polygon vertex, the additional length is $O(\delta n)$. Hence, triangulating $\mathcal{D}$ increases the weight by an additive term $O(\delta n)$.

There is a subtle problem when holes are allowed to degenerate to line segments. When this happens, the triangles on opposite sides of a line segment may not be compatible with each other. Thus, when generating Steiner triangulation, we can only accommodate holes that are either simple polygons or isolated points.

Under this assumption, we can now follow the approach in [5] to show that the triangulation of the $\mathcal{D}$ approximates the MWST when δ is sufficiently small (e.g. $\mathcal{D}$ is untruncated).

Theorem 1. *Given a polygon P of n vertices, the weight of the triangulation of $\mathcal{D}$ is bounded by $O(w(T) + \delta n)$, where $w(T)$ is the weight of any Steiner triangulation T of P.*

Proof: We introduce the Steiner points of T as degenerate holes. T can then be viewed as the MWT of the new polygon. If we apply our decomposition algorithm on the new polygon, we will obtain a refinement of $\mathcal{D}$ that has a larger total length than $\mathcal{D}$. Lemma 3 shows the total length of the refined decomposition is bounded by $O(w(T))$. So the triangulation of $\mathcal{D}$ has weight $O(w(T)+\delta n)$ as discussed before. ◻

If we set $\delta = R/n$, where R is the width of the smallest enclosing square of P, then the triangulation of $\mathcal{D}$ still approximates the MWST as $w(T) \geq R$ for any Steiner triangulation T. The advantage of this setting of δ is that $\mathcal{D}$ can be computed efficiently as proved below.

Theorem 2. *Given a polygon of n vertices, a Steiner triangulation of $O(n \log n)$ size can be computed in $O(n \log n)$ time which approximates the MWST to within a constant factor.*

Proof: We set $\delta = R/n$. Then the quadtree clearly has $O(\log n)$ levels. Since the work done per level of the quadtree is bounded by the complexity of cells created and crowded subpolygons at that level, we bound this quantity below. Consider one level of the quadtree. Let b be a box at this level. Let α be a crowded subpolygon of b or a cell created at b. Our approach is to charge non-polygon vertices of α to some nearby polygon vertices.

Case 1: α contains some polygon vertex. Then we charge its non-polygon vertices to its polygon vertices. Each polygon vertex is charged at most ten times this way.

Case 2: α is a cell without any polygon vertex and α is bordered by a polygon edge h with an endpoint w in $3p(b)$. Then we charge the vertices of α (at most 8) to w. Observe that α is within a distance of $6\mathit{size}(b)$ from w. Since h borders no more than $12^2 = 144$ such cells within a distance of $6\mathit{size}(b)$ from w, w is charged at most $144 \times 8 = 1152$ times this way.

Case 3: α is a cell without any polygon vertex and the endpoints of each polygon edge bordering α are outside $3p(b)$. Let α' be the subpolygon of $p(b)$ that contains α. Since α' was split, $\mathit{zone}(\alpha')$ contains a polygon vertex. Thus, $\mathit{zone}(\alpha')$ has some polygon vertex v that can see α. We charge the vertices of α to v. We can similarly argue as in case 2 that v is charged at most $144 \times 8 = 1152$ times this way.

Combining cases 1, 2 and 3, we conclude that the complexity of cells created and crowded subpolygons is $O(n)$ at each level of the quadtree. Hence, the results follow. ◻

5 Ray Shooting

5.1 Average Case

We briefly explain the average-case model given in [1] and point out why it is better to triangulate the interior of P alone when the exterior of polygon is inaccessible to light. Recall that the average is taken over all random choices of light source on the boundary of P and over all random choices of ray direction.

We first relax to talk about any planar subdivision $\mathcal{S}$ (not necessarily a triangulation) of the interior of P. We look at the *line stabbing query*: given a query line ℓ, report the boundary edges of P intersected by ℓ. For line ℓ, let $o(\ell)$ and $t(\ell)$ be the number of boundary edges of P and edges of $\mathcal{S}$ intersected by ℓ respectively. Let μ be the standard measure on lines invariant under rigid motions. Let $L(U)$ be the set of lines that intersect a set of line segments U. Then for a query line, the average number of subdivision edges intersected per boundary edge intersected is given by $\int t(\ell)d\mu / \int o(\ell)d\mu$. Denote this ratio by $\Gamma(\mathcal{S})$. One can interpret the ratio as: if one starts from a boundary edge, then one expects to encounter an average of $\Gamma(\mathcal{S})$ subdivision edges before exiting the interior of P. This is exactly the average-case ray shooting complexity.

Standard result in integral geometry states that for any line segment e, $\mu(L(e))$ is two times the length of e. Therefore, $\Gamma(\mathcal{S})$ can be rewritten as the ratio of the total length of $\mathcal{S}$ to the perimeter of P. Hence, it is desirable to obtain a planar subdivision $\mathcal{S}$ with minimum total length. For ray shooting purposes, one must be able to determine what is the next face to visit efficiently. Therefore, it is often required that each face in $\mathcal{S}$ has at most a constant number of adjacent faces. A Steiner triangulation of P is a natural choice.

How does our triangulation compare with the triangulation in [1]? The first step in [1] is to compute a quadtree decomposition of the vertices of P. Then this decomposition will basically be triangulated to produce a Steiner triangulation. Figure 3 shows that the quadtree decomposition of the vertices may already have total length $\Omega(\log n)$ away from some Steiner triangulation of P. Thus, it is advantageous to triangulate the interior of P alone.

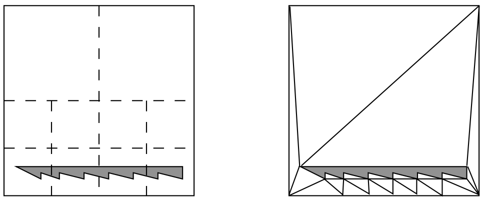


Fig. 3. In the left figure, the dense row of vertices on the hole forces the quadtree decomposition to add horizontal line segments across the polygon as many as $\Omega(\log n)$ times. Thus, the width of the polygon will be added $\Omega(\log n)$ times. In contrast, as shown in the right figure, one can triangulate the polygon interior so that the width of the polygon is also added a constant number of times.

5.2 Query-Sensitive Ray Shooting

Given a query ray r, a planar subdivision and an accompanying data structure is described in [7] that answers ray shooting in $O(\log n + cscc(r))$ time, where $cscc(r)$ is the minimum number of ϵ-strongly simple C-balls that cover r up to the first intersection. The space required is $O(n)$. In the following, we show that the Steiner triangulation resulting from the untruncated $\mathcal{D}$ also supports query-sensitive ray shooting.

Lemma 4. *Let B be an ϵ-strongly simple C-ball of radius s. If B intersects a cell in the untruncated $\mathcal{D}$ of size s', then $s' \geq \epsilon s/4\sqrt{2}$.*

Proof: Let γ be any cell intersected by B in the untruncated $\mathcal{D}$. Assume that to the contrary that $size(\gamma) < \epsilon s/4\sqrt{2}$. Let α be the subpolygon of the parent of the home box of γ that contains γ. Since $size(\alpha) < \epsilon s/2\sqrt{2}$ and α intersects B, $zone(\alpha)$ lies completely within $(1+\epsilon)B$. Since $(1+\epsilon)B$ is a simple ball, $zone(\alpha)$ contains at most one polygon vertex. Thus, α would not be crowded, contradiction. ⊔⊓

Lemma 4 implies that the untruncated $\mathcal{D}$, and hence its triangulation, has $O(cscc(P))$ size. Similarly, for any ray r, the number of triangles traversed by r up to the first intersection is $O(cscc(r))$. One can locate the cell containing the light source in $O(\log cscc(P))$ time using point location.

Theorem 3. *Given a polygon of n vertices, there is an approximate MWST of $O(cscc(P))$ size such that given a query ray r, the ray shooting query can be answered using the approximate MWST in $O(\log cscc(P) + cscc(r))$ time.*

As argued in [4,7], $cscc(P)$ is expected to be small in practice (e.g. $O(n)$) and so the query time in Theorem 3 may often be good enough. When $cscc(P)$ is actually large, we can alternatively locate the light source by descending the quadtree. This gives a running time of $O(\log(R/r)\log n)$, where R is the width of the minimum enclosing square of P, and r is the size of the largest ϵ-strongly C-simple ball containing the light source. When the local geometry around the light source is very simple, one expects $O(\log(R/r))$ to be small.

References

1. B. Aronov and S. Fortune, Average-case ray shooting and minimum weight triangulation, in *Proc. 13th ACM Symposium on Computational Geometry*, 204–211, 1997. 367, 368, 374, 375
2. S. Arya, S.W. Cheng, and D.M. Mount, Approximation algorithms for multiple-tool milling, *Proc. 14th ACM Symposium on Computational Geometry* 1998, 297–306. 368
3. M. Bern, D. Eppstein, and J. Gilbert, Provably good mesh generation, *Journal of Computer and System Sciences*, 48 (1994) 384–409. 371
4. M. de Berg, M. Katz, A.F. van der Stappen, J. Vleugels, Realistic input models for geometric algorithms, in *Proc. 13th ACM Symposium on Computational Geometry*, 294–303, 1997. 376
5. D. Eppstein, Approximating the minimum weight Steiner triangulation, *Discrete & Computational Geometry*, 11 (1994) 163–191. 367, 371, 372, 373
6. K.Y. Fung, T.M. Nicholl, R.E. Tarjan, and C.J. Van Wyk, Simplified linear-time Jordan sorting and polygon clipping. *Information Processing Letters*, 35 (1990) 85–92. 370, 371

7. J.S.B. Mitchell, D.M. Mount, and S. Suri, Query-sensitive ray shooting, *International Journal of Computational Geometry & Applications*, 7 (1997) 317–347. 368, 371, 375, 376
8. F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985. 370

Optimality and Integer Programming Formulations of Triangulations in General Dimension

Akira Tajima*

Tokyo Research Laboratory, IBM Research
tajima@trl.ibm.co.jp

Abstract. The properties of triangulations in two and three dimensions are computationally investigated by using integer programming (IP). Three IP formulations of triangulations are introduced, two based on the stable set problem, and the other based on the set partitioning problem. Some properties that are interesting from a theoretical or practical point of view are considered as objective functions for IP. Finally, some computational results are given. This approach allows three-dimensional triangulations to be treated in a flexible and efficient way, and has led to the discovery of some interesting properties of three-dimensional triangulations.

1 Introduction

Triangulation is a major topic in computational geometry, and there have been many studies of various approaches to triangulating a configuration of points, such as Delaunay triangulation, the minimum weight triangulation, and the lexicographic triangulation. Most of these studies have focused on triangulations in two dimensions, and less attention has been paid to triangulations in higher dimensions.

As a result of the recent advances in the performance of computers, however, the number of applications using triangulation in three dimensions is growing. For the finite element method (FEM), a three-dimensional polyhedron has to be divided into meshes. There are many meshing techniques: one approach is to distribute points inside the polyhedron and generate meshes by triangulating them. This approach is gaining popularity because of its simplicity and independence of dimensions [14]. Another major application is volume rendering, which is a method for visualizing the results of FEM, or semi-transparent objects such as clouds and flames. Unlike ordinary computer graphics methods that visualize the surface of materials, volume rendering meshes the three-dimensional space [15].

To contribute to these applications, the properties of triangulations in three dimensions should be further investigated. In three dimensions, some properties that are valid in two dimensions do not hold. Bistellar flips (Figure 1) show

* also belongs to Department of Information Science, University of Tokyo

that the number of triangles is not constant in three dimensions. Schönhardt's polytope (Figure 2), which can be obtained by twisting a prism in such a way that the side faces become concave, shows that not all polyhedra can be triangulated in three dimensions. Some results on triangulatability are given in [13].

In two dimensions, Delaunay triangulation is optimal in many respects; for example, it maximizes the minimum angle, and minimizes the largest circumsphere and the largest minimum enclosing sphere. Many applications, such as meshing algorithms for FEM, depend on this characteristic. But in three dimensions, it only minimizes the largest minimum enclosing sphere [12], and does not necessarily lead to optimality in other respects. In other words, Delaunay triangulation is not so good in three or more dimensions. Thus, it is necessary to obtain a good triangulation for each individual application, namely, to optimize the triangulation.

In this paper we investigate the optimality of triangulations in two and three dimensions by using integer programming (IP) formulations. We first give three formulations in general dimension. Two of them are based on the stable set problem, and the other is based on the set partitioning problem. We then compare these formulations in size, and in other respects related to the efficiency of the procedure for solving the problem. We consider several objective functions in both two and three dimensions, which are interesting from a theoretical or practical point of view. We also give some computational results. Through the experiments, we found some interesting properties of triangulations in three dimensions.

As a preliminary step, we define some special terms. A *d-simplex* is a d-dimensional polytope that is the convex hull of $d+1$ affinely independent points. For example, a line segment, a triangle, and a tetrahedron correspond to a 1-simplex, a 2-simplex, and a 3-simplex, respectively. An *i-face* of a d-simplex is an i-simplex ($0 \leq i \leq d$) that is the convex hull of a subset of the vertices of the d-simplex. In particular, a $(d-1)$-face is called a *facet*. Two d-simplices *intersect* when the intersection is non-empty and is not a face of at least one of the two simplices. In this paper, especially for IP formulations, we consider the division of the convex hull of a point configuration $\mathcal{A}$ of n points in d-dimensional space using d-simplices, but we use the term *triangulation* for convenience. We assume that $\mathcal{A}$ is a configuration in general position (no $d+1$ points lie on the same $(d-1)$-dimensional hyperplane).

This paper is organized as follows. In Section 2, we give two kinds of IP formulations of triangulations in general dimension, then compare and investigate them. In Section 3, we consider objective functions. In Section 4, we give some computational results. Section 5 summarizes this paper.

2 Formulations

In this section, we give IP formulations by regarding triangulation as the stable set problem (Formulations 0 and 1), or the set partitioning problem (Formulation 2). Especially for the former, we consider two cases: that in which

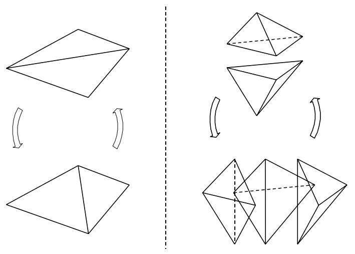


Fig. 1. Bistellar flips in 2 and 3 dimensions

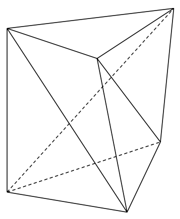


Fig. 2. Schönhardt's polytope

each d-simplex corresponds to an element (Formulation 0), and that in which each i-simplex ($0 \leq i \leq d$) corresponds to an element (Formulation 1).

2.1 Formulation 0: The Stable Set Problem of d-simplices

In the intersection graph $G(V, E)$ of d-simplices, V corresponds to the set of d-simplices, and an arc is defined between two nodes when the corresponding two d-simplices intersect. A triangulation has no pair of simplices that intersect and thus corresponds to a stable set; further, it is maximal, because another d-simplex surely intersects with some d-simplices that constitute the triangulation. On the basis of this observation, Imai [10] gave an algorithm for enumerating triangulations in general dimension.

As mentioned in Section 1, not all polytopes can be triangulated in dimensions higher than two. Thus, there exist cases in which regions like Schönhardt's polytope remain untriangulated inside the convex hull, and the maximal stable set is not a triangulation. Therefore, we have to ensure that the stable set becomes a triangulation by imposing the condition that the sum of the volume v_i of d-simplex i is equal to the volume V of the convex hull.

$$\textbf{minimize} \sum_i c_i x_i \tag{1}$$

s.t.

$$\begin{aligned} &x_i \in \{0, 1\} \\ &x_i + x_j \leq 1 \text{ (for } i, j \text{ s.t. } d\text{-simplex } i \text{ and } d\text{-simplex } j \text{ intersect)} \\ &\sum_i v_i x_i = V \end{aligned}$$

Here, checking the intersections among d-simplices is essentially redundant. For example, consider triangulations of the convex hull of four points on a plane. We have four triangles, and four pairs of triangles that intersect. On the other hand, the four intersections are derived from an intersection of two diagonals. In other words, if we explicitly handle lower-dimensional faces of d-simplices, the formulation can be more efficient.

2.2 Formulation 1: The Stable Set Problem of i-simplices ($i \leq d$)

In the two-dimensional case, Kyoda et al. [11] formulated the minimum weight triangulation (MWT) problem as the stable set problem on the intersection graph of edges (1-simplices). MWT is a famous problem for which it is not known whether a solution can be obtained in polynomial time [9]. In two dimensions, the maximal stable sets of edges are always maximum, and the number of edges M is constant. Thus MWT is obtained by using the following formulation:

$$\textbf{minimize} \sum_i c_i x_i \tag{2}$$

s.t.

$$\begin{aligned} x_i &\in \{0,1\} \\ x_i + x_j &\leq 1 \text{ (for } i,j \text{ s.t. edge } i \text{ and edge } j \text{ intersect)} \\ \sum_i x_i &= M \end{aligned}$$

Extending to Higher Dimensions The above formulation (2) is valid only in two dimensions, and the cost and variables are assigned only to edges, so we need to extend it to higher dimensions and assign variables to d-simplices. We first consider the simplest example in three dimensions, a bistellar flip defined on the convex hull of five points. In the lattice-like structure in Figure 3, the i-th layer corresponds to $(i-1)$-simplices, and straight arcs among layers correspond to face relations of simplices in different dimensions. For example, edges (0,1), (0,4), and (1,4) are facets of, and necessary for triangle (0,1,4). Thus, the lattice is interpreted as a poset when we assign a variable to each simplex in such a way that the variable is equal to 1 if the simplex appears in the triangulation, and 0 otherwise. We will call the relations defined by the poset *lattice constraints.*

The lattice itself is independent of the configuration, whereas information on intersections depends on the configuration. In Figure 3, edge (0,4) and triangle (1,2,3) intersect. On the other hand, point (4) and tetrahedron (0,1,2,3) intersect in Figure 4.

When edge (0,4) and triangle (1,2,3) intersect, only one of them can appear in the triangulation. This can be represented as a constraint that the sum of the corresponding variables is less than or equal to 1. The following condition is necessary to make the formulation efficient:

Proposition 1. *All the vertices of simplices are assumed to be in general position. Then, two d-simplices intersect if and only if they have a k-face and a j-face, respectively, that intersect and satisfy $k + j = d$.*

We obtain a set of d-simplices that do not intersect by imposing the intersection constraints only on the pairs of simplices whose sum of dimensions is equal to d, and the lattice constraints. The volume constraint is again necessary, as in formulation (1), for the set to be a triangulation.

$$\textbf{minimize} \sum_i c_i x_i^d \tag{3}$$

s.t.

$$x_i^k \in \{0,1\} \ (0 \le k \le d)$$
$$x_i^k + x_j^{d-k} \le 1 \ (k\text{-simplex } i \text{ and } (d-k)\text{-simplex } j \text{ intersect},\ 0 \le k \le d)$$
$$x_i^k - x_j^{k+1} \ge 0 \ (k\text{-simplex } i \text{ is a facet of } (k+1)\text{-simplex } j,\ 0 \le k \le d-1)$$
$$\sum_i v_i x_i^d = V$$

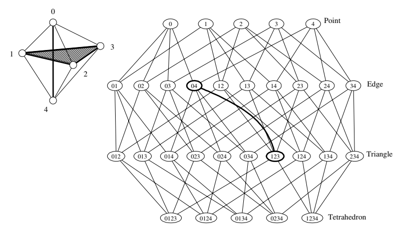


Fig. 3. Face relations among simplices

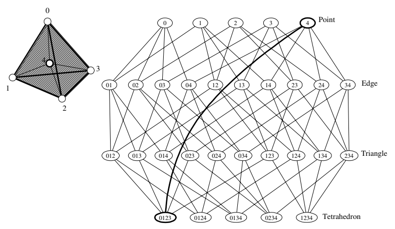


Fig. 4. Intersection of a 0-simplex and a 3-simplex

2.3 Formulation 2: The Set Partitioning Problem

We use the term *chambers* for the minimal cells obtained by dividing the convex hull with all the possible $(d-1)$-simplices. Obtaining a triangulation can be treated as an instance of the set partitioning problem by regarding each chamber as an element, and each d-simplex as a subset [1].

$$\textbf{minimize}\ cx^d \tag{4}$$

s.t.

$$x_i^d \in \{0,1\}$$
$$Ax^d = 1 \quad A_{ij} = \begin{cases} 1 & d\text{-simplex } j \text{ contains chamber } i \\ 0 & \text{otherwise} \end{cases}$$

Cocircuit Form Constraints A large amount of geometric computation is necessary for handling chambers, and therefore we consider another type of constraint called *cocircuit form* constraints instead. Let e_k be a $(d-1)$-simplex. Two half-spaces $\mathcal{H}^+_{e_k}, \mathcal{H}^-_{e_k}$ are defined by a hyperplane $\mathcal{H}_{e_k}$ containing e_k. Let $e_k \cup \{a\}$ be a d-simplex t_i defined by an element a of the point set $\mathcal{A}$ and e_k. The following constraint holds for all $(d-1)$-simplices [7].

$$\sum_{t_i = e_k \cup \{a\}, a \in \mathcal{A} \cap \mathcal{H}^+_{e_k}} x_i - \sum_{t_i = e_k \cup \{a\}, a \in \mathcal{A} \cap \mathcal{H}^-_{e_k}} x_i = \begin{cases} \pm 1 & e_k \text{ is on the boundary} \\ & \quad \text{of the convex hull} \\ 0 & \text{otherwise} \end{cases}$$

For $(d-1)$-simplices on the boundary, the above equation is the same as the set partitioning constraint for the adjacent chamber. For interior $(d-1)$-simplices, it corresponds to the difference of the constraints for the chambers on both sides.

De Loera et al. [7] showed that the cocircuit form constraints are sufficient to define the affine hull of the characteristic vectors of d-simplices that form a triangulation. From now on, we will call the formulation using the cocircuit form constraints *Formulation 2*.

2.4 Comparison of Formulations

In this section, we compare Formulations 0, 1, and 2. Table 1 shows the size of each formulation. Although Formulation 1 uses fewer constraints than Formulation 0, Formulation 2 is still the most compact.

Table 1. Size of each formulation

	#variables	# constraints	# v/c
Formulation 0	$O(n^{d+1})$	$O(n^{2(d+1)})$	$O(1)$
Formulation 1	$O(n^{d+1})$	$O(n^{d+2})$	$O(1)$
Formulation 2	$O(n^{d+1})$	$O(n^{d})$	$O(n)$

#variables: total number of variables
#constraints: total number of constraints
#v/c: number of variables in a constraint

Polytopes We denote the polytope that corresponds to the linear programming relaxation of formulation 0, 1, and 2 by P_0, P_1', and P_2 respectively. The projection of P_1' to the subspace that corresponds to the variables for d-simplices is denoted by P_1. Then, the following relations hold among these polytopes:

Proposition 2. $P_0 = P_1$.

Lemma 1. *For the convex hull of $d+3$ points in d dimensions, there exist two triangulations that share no d-simplex. Further, there exists a d-simplex that lies inside the convex hull and does not belong to either of the two triangulations.*

Proposition 3. *$P_2 \subset P_0$ $(n \geq d+3)$.*

Proof outline: By focusing on a chamber, we can observe that both the volume and intersection constraints are satisfied if the set partitioning condition is satisfied. Thus $P_2 \subseteq P_0$.

From Lemma 1, we can configure the overlap of two triangulations, both of which are assigned a weight of $\frac{1}{2}$. We then perturb it slightly by using another simplex so that it still satisfies the constraints of Formulation 0. It then violates some chamber constraints, and $P_2 \subset P_0$. □

Spanning Triangulations All the formulations above allow triangulations that do not use points inside the convex hull. However, triangulations are often assumed to use all the points: we call them spanning triangulations. In order to avoid non-spanning triangulations, we eliminate non-empty d-simplices in advance. For random point sets, the expected number of empty d-simplices is $O(n^d)$ [3], which is significantly smaller than the upper bound $\binom{n}{d+1}$ achieved when all the points are on the boundary of the convex hull. The lower bound is $\binom{n}{d-1}$ [3].

2.5 Cutting Planes

As cutting planes for the problem of formulation (2), Kyoda et al. [11] applied clique cuts and odd-cycle cuts, both of which are well known for the stable set problem, and convex polygon cuts which use geometric information on triangulations. Formulations 0 and 1 are based on the stable set problem, and these cutting planes can cut off fractional solutions of their linear programming relaxations. In this section, we investigate the effectiveness of cutting planes with respect to Formulation 2.

The convex polygon cut is based on the property that the number of edges is constant for triangulations in two dimensions. Thus, this cut is valid only in two dimensions. Although it was given as a constraint on the number of edges, it can also be described as a constraint on the number of triangles, as follows:

Definition 1. *Let $\mathcal{V}$ denote a configuration of points, C denote its convex hull, and M denote the cardinality of the spanning triangulations of $\mathcal{V}$. Further let x denote the incident vector of a spanning triangulation of a point configuration $\mathcal{A}$ $(\mathcal{A} \supseteq \mathcal{V}, \mathcal{A}\backslash\mathcal{V}\cap\mathcal{C} = \emptyset)$, and let V denote the dimensions of x, which correspond to triangles that only have the elements of $\mathcal{V}$ as their vertices. The following inequality holds:*

$$\textit{Convex polygon cut} \quad \sum_{i\in V} x_i \leq M$$

Here we focus on an example in which Formulation 2 has a fractional solution [7]. A set of vertices $1, \ldots, 5$ of a regular pentagon and its center 0 constitutes a point set in two dimensions, and has 20 triangles and 16 triangulations. The vector $x \in \mathcal{R}^{20}$ with coordinates $x_{\{123\}} = x_{\{234\}} = x_{\{345\}} = x_{\{145\}} = x_{\{125\}} =$

$x_{\{013\}} = x_{\{024\}} = x_{\{035\}} = x_{\{014\}} = x_{\{025\}} = \frac{1}{2}$, and all other coordinates $= 0$, satisfies the constraints of Formulation 2. The subgraph of the intersection graph of triangles induced by the nodes corresponding to the triangles with value $\frac{1}{2}$ and without vertex 0, constitutes an odd-cycle of size 5. The corresponding constraint

$$x_{\{123\}} + x_{\{234\}} + x_{\{345\}} + x_{\{145\}} + x_{\{125\}} \leq 2$$

cuts off x. Thus, odd-cycle cuts can be effective cutting planes. On the other hand, the convex polygon cut $\sum_{ijk} x_{ijk} \leq M(= 5)$ is satisfied by x and redundant. For convex polygon cuts, we state the following:

Conjecture 1. Any convex polygon cut is valid for P_2.

3 Objective Functions

In this section, we introduce some objective functions that are interesting from a theoretical or practical point of view. Objective functions can be categorized into two types. One is simple summation, which only requires us to solve IP on the basis of one of the formulations we introduced in Section 2. The other is bottleneck optimization, such as minimizing the maximum. If we try to solve such problems solely by using IP, the branch-and-bound procedure just goes progressively deeper and the lower bound never improves, and thus we cannot obtain a solution. Binary search is one way to avoid this numerical instability.

Minimize the maximum aspect ratio. In applications such as FEM, we should avoid flatness of triangles for the sake of computational stability and accuracy. The most straightforward index of the flatness is the aspect ratio (AR), which is the ratio of the radii of the circumscribing and inscribing spheres.

Maximize the minimum angle. Delaunay triangulation in two dimensions maximizes the minimum angle. In three dimensions, the angle itself has two varieties; the dihedral angle and the solid angle. There are several meshing algorithms that avoid some of the bad – too large or too small – angles (see [4], for example), but none for triangulating a point configuration. We can obtain a solution by specifying the min-max/max-min angle as the objective function.

Minimize the number of triangles. The number of triangles varies in three dimensions, and is an interesting objective function related to the following open problem:

Open Problem 1 *Is there a polynomial-time algorithm for triangulating an arbitrary convex polyhedron with the minimum number of tetrahedra ? [5]*

Minimize the weight. MWT in two dimensions is an interesting problem, as we mentioned in Section 2.2. In three dimensions, the weight is extended from the edge length to the surface area of the triangle.

4 Computational Experiments

This section gives the results of computational experiments based on Formulation 2. All the experiments were done on an IBM RS/6000 model 990, using the IBM Optimization Subroutine Library to solve mathematical programming problems. We also used the library CGAL [6] for handling geometric objects, and for obtaining Delaunay triangulation, we used the program Triangle [16] for two dimensions, and the program DeWall [8] for three dimensions.

4.1 Problem Size and Computational Time

We measured the size and the CPU time for obtaining the minimum weight (spanning) triangulation of uniformly distributed point sets in two and three dimensions (Tables 2 and 3). The size of solvable problems is quite small in three dimensions. Further, if the point set is in convex position, all the triangles/tetrahedra are non-empty and the number of columns becomes larger, and the size of solvable instances becomes smaller.

We obtained integer solutions by means of the linear programming relaxations in all the cases, whereas in [11], based on the stable set problem, we obtained fractional solutions in many cases and applied cutting planes and the branch-and-bound method.

We experimented with several kinds of objective functions, including the ones in Section 3, and encountered fractional solutions after solving the relaxed linear programming problems only when the objective function was the cardinality, that is, when the problem was unweighted. Formulation 2 is fairly good practically, but still, the cutting planes we mentioned in Section 2.5 would be necessary to solve unweighted cases efficiently.

Table 2. Size and CPU time of the minimum weight triangulation in 2D

No. of points	10	20	30	40	80	160	240	320
No. of rows	45	190	435	780	3160	12720	28680	51040
No. of columns	71	482	1220	2328	11209	47837	109998	197883
IP (sec.)	0.24	0.72	1.78	4.25	45.91	727.99	3956.20	13801.49
Others (sec.)	0.31	1.89	6.21	14.24	131.35	1345.19	5283.15	13770.27

Table 3. Size and CPU time of the minimum weight triangulation in 3D

No. of points	10	20	30	40	50
No. of rows	120	1140	4060	9880	19600
No. of columns	194	3771	19638	57735	139982
IP (sec.)	0.42	9.08	148.27	6374.83	66798.32
Others (sec.)	0.69	10.29	48.67	145.28	372.84

IP: CPU time for solving the IP problem
Others: CPU time for the rest

4.2 Optimality

Figures 5, 6, and 7 show the cardinality, weight and maximum aspect ratio, respectively, of triangulations in three dimensions. We used 10 point sets of cardinality 10, 20, and 30 randomly generated in a unit cube with different seeds. D, W, C, and AR in these figures represent Delaunay triangulation, the minimum weight triangulation, the minimum cardinality triangulation, and the triangulation with the minimum maximum aspect ratio, respectively.

From Figure 5 we can observe that Delaunay triangulation tends to contain more tetrahedra than the other triangulations. Figure 6 shows that Delaunay triangulation is not at all close to the minimum weight triangulation in three dimensions. Figure 7 shows that Delaunay triangulation avoids flat tetrahedra. There is still a gap between Delaunay triangulation and the optimal solution, which is obtained by the triangulation with the minimum maximum aspect ratio.

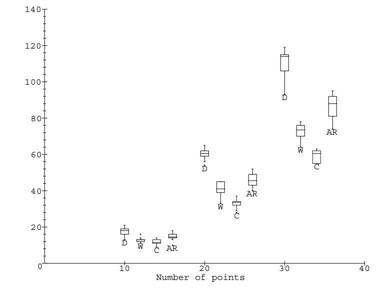


Fig. 5. Cardinality of triangulations

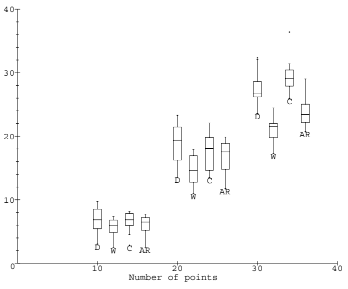


Fig. 6. Weight of triangulations

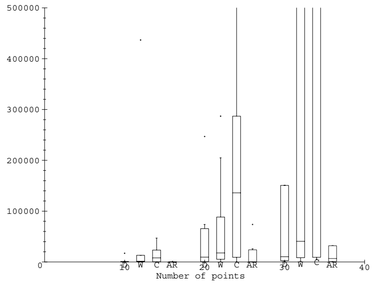


Fig. 7. Minimum maximum aspect ratio of triangulations

5 Conclusions and Remarks

In this paper, we have introduced IP formulations of triangulations of a point configuration in general dimension, and shown through computational experiments that the properties of triangulations differ significantly between two and three dimensions.

Another very important property of triangulations in three dimensions is that not all polyhedra can be triangulated [13]. Through the use of IP and maximization of the volume that can be triangulated, it may be possible to gain some insight into triangulatability. Among the formulations introduced in this paper, the one using cocircuit form constraints was the most efficient for triangulations of a point configuration. But it is still an open question which formulation is the most efficient for triangulations of non-convex polytopes.

Our experiments were based on linear programming, and it would be interesting to compare our approach with graph-based algorithms for the stable set problem ([2] for example).

References

1. T. V. Alekseyevskaya. Combinatorial bases in systems of simplices and chambers. *Discrete Mathematics*, 157:15–37, 1996. 381
2. E. Balas and J. Xue. Minimum weighted coloring of triangulated graphs, with application to maximum weighted vertex packing and clique finding in arbitrary graphs. *SIAM Journal of Computing*, 20(2):209–221, 1991. 387
3. I. Bárány and Z. Füredi. Empty simplices in Euclidian spaces. *Canad. Math. Bull.*, 30:436–445, 1987. 383
4. M. Bern, P. Chew, D. Eppstein, and J. Ruppert. Dihedral bounds for mesh generation in high dimensions. In *6th ACM-SIAM Symp. Discrete Algorithms*, pages 189–196, 1995. 384
5. M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 47–123. World Scientific, 2nd edition, 1995. 384
6. CGAL. http://www.cs.ruu.nl/CGAL/. 385
7. J. A. De Loera, S. Hosten, F. Santos, and B. Sturmfels. The polytope of all triangulations of a point configuration. *Documenta Mathematica*, 1:103–119, 1996. 382, 383
8. DeWall. http://miles.cnuce.cnr.it/cg/swOnTheWeb.html. 385
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. 380
10. H. Imai and K. Imai. Triangulation and convex polytopes. In *RIMS Kokyuroku*, pages 149–166. Research Institute for Mathematical Sciences, Kyoto University, 1996. (in Japanese). 379
11. Y. Kyoda, K. Imai, F. Takeuchi, and A. Tajima. A branch-and-cut approach for minimum weight triangulation. In *Proceedings of the 8th Annual International Symposium on Algorithms and Computation (ISAAC '97)*, volume 1350 of *Lecture Notes in Computer Science*, pages 384–393. Springer Verlag, 1997. 380, 383, 385
12. V. T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proc. 7th ACM Symp. Computational Geometry*, pages 357–363, 1991. 378
13. J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete & Computational Geometry*, 7:227–253, 1992. 378, 387
14. K. Shimada. Physically-based automatic mesh generation. *Simulation*, 12(1):11–20, 1993. (in Japanese). 377
15. P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990. 377
16. Triangle. http://www.cs.cmu.edu/~quake/triangle.research.html. 385

Space-Efficient Approximation Algorithms for MAXCUT and COLORING Semidefinite Programs

Philip N. Klein[1] and Hsueh-I Lu[2]

[1] Department of Computer Science, Brown University, USA
klein@cs.brown.edu

[2] Department of Computer Science and Information Engineering
National Chung-Cheng University, Taiwan
hil@cs.ccu.edu.tw

Abstract. The essential part of the best known approximation algorithm for graph MAXCUT is approximately solving MAXCUT's semidefinite relaxation. For a graph with n nodes and m edges, previous work on solving its semidefinite relaxation for MAXCUT requires space $\tilde{O}(n^2)$. Under the assumption of exact arithmetic, we show how an approximate solution can be found in space $O(m + n^{1.5})$, where $O(m)$ comes from the input; and therefore reduce the space required by the best known approximation algorithm for graph MAXCUT.

Using the above space-efficient algorithm as a subroutine, we show an approximate solution for COLORING's semidefinite relaxation can be found in space $O(m) + \tilde{O}(n^{1.5})$. This reduces not only the space required by the best known approximation algorithm for graph COLORING, but also the space required by the only known polynomial-time algorithm for finding a maximum clique in a perfect graph.

1 Introduction

Semidefinite programming [35] is the mathematical programming for optimizing a linear function of a matrix X subject to linear constraints and the constraint that X is symmetric and positive semidefinite. (The eigenvalues of a symmetric matrix are all real. A symmetric matrix is positive semidefinite if all of its eigenvalues are nonnegative.) It is a special case of convex programming, and a generalization of linear programming. Its application on combinatorial optimization problems was pioneered by Lovász's work on the Shannon capacity of a graph, which is also known as the theta function [25]. The polynomial-time solvability of semidefinite programs leads to the only known polynomial-time algorithms for some optimization problems for perfect graphs, such as MAXCLIQUE (and therefore MAX STABLE SET) [13], and COLORING [14].

Recently semidefinite programming is emerging as an important technique for designing approximation algorithms. Goemans and Williamson [12] gave an approximation algorithm for graph MAXCUT, whose approximation ratio is significantly better than that of the previously known algorithms. The essential part of their algorithm is obtaining a near-optimal solution to a semidefinite program.

Based on this work, Karger, Motwani, and Sudan [19] discovered the best known approximation algorithm for coloring a k-colorable graph. Besides MAX-CUT and COLORING, the technique of using semidefinite programming has been successful in designing approximation algorithms for many other optimization problems [4,12,9,12,9,20,11,2,7,3]. Each of these improved algorithms is based on obtaining a near-optimal solution to a semidefinite program, which is referred as the *semidefinite relaxation* of the underlining optimization problem. Therefore how to approximately solve these semidefinite relaxations efficiently, both in time and space, is practically important.

The first algorithm proposed for solving semidefinite programs was based on the ellipsoid method [13]. Nesterov and Nemirovsky showed [27] how to use the interior-point methods to solve semidefinite programs. Alizadeh [1] showed how an interior-point algorithm for linear programming could be directly generalized to handle semidefinite programming. Since the work of Alizadeh, there has been a great deal of research into such algorithms [18,36,31,37,15,24,10]. A simplex-type method was also discovered by Pataki [29]. Among these varieties, the performance of interior-point methods are the best in both theory and practice. Suppose there are ℓ constraints in a semidefinite program, whose variable is an $n \times n$ matrix. A near-optimal solution for the semidefinite program can be obtained in time $\tilde{O}(\sqrt{n}\ell^3)$ using interior-point methods. For example, the time required for interior-point methods to find near-optimal solutions for MAX-CUT's and COLORING's semidefinite relaxations are $\tilde{O}(n^{3.5})$ and $\tilde{O}(\sqrt{n}m^3)$, respectively, where n is the number nodes and m is the number of edges.

Recently we [22] show how to obtain an approximate solution for some semidefinite relaxations more quickly by exploiting the structure of the underlining optimization problems. Specifically we show how to obtain near-optimal solutions for MAXCUT's and COLORING's semidefinite relaxations in time $\tilde{O}(mn)$, where n is the number of nodes and m is the number of edges.

In this paper we explore the efficiency of algorithms for semidefinite relaxations in another domain, i.e. the space complexity. The space required by interior-point methods is $\Omega(n^2)$ because it has to maintain a full-rank $n \times n$ matrix during its execution. Our algorithm for MAXCUT's semidefinite relaxation given in [22] is potentially better because it does not have to maintain a full-rank matrix all the time. Specifically, it starts with an initial rank-one matrix, and the proceeds iteratively. In each iteration it apply a rank-one update to the current solution matrix. Unfortunately the number of iterations can be as large as n. Therefore the output tends to be full-rank, and its space complexity is thus asymptotically the same as that of interior-point methods. Similarly the output of our previous algorithm for COLORING's semidefinite relaxation tends to have full rank.

In this paper we show how to equip our previous time-efficient algorithm for MAXCUT with a *rank-reduction procedure*. The purpose of the rank-reduction procedure is to keep the rank of the current solution matrix at $O(\sqrt{n})$. As a result, the space required by the new algorithm during the execution can be kept at $(n^{1.5})$. The space-efficient algorithm for MAXCUT's semidefinite relaxation can be applied to solving COLORING's semidefinite program, as shown in our previous paper [22], and results in a space-efficient approximation algorithm for COLORING's semidefinite relaxation. The space-efficient algorithm for COLORING's semidefinite program can then be applied to the only known polynomial-time algorithm for MAXCLIQUE on perfect graphs, and results in a space-efficient algorithm for finding a maximum clique in a perfect graph.

2 Overview

Space efficiency is a practical issue in solving mathematical programs. Usually the program for solving a mathematical program has to keep everything in the main memory. Therefore the $O(n^2)$ space requirement for solving semidefinite programs has been keeping the size of a practically solvable problem small, even if we are willing to spend more time. In this paper we show how to obtain solutions for three problems in space $\tilde{O}(m + n^{1.5})$. Our results could increase the size of practically solvable problem by a significant factor. For example, let the size of main memory be M. When $m = O(n^{1.5})$, the solvable size n will be increased by a factor of $M^{\frac{2}{3}-\frac{1}{2}} = M^{\frac{1}{6}}$ using our space-efficient algorithms

MAXCUT. Our previous time-efficient algorithm for MAXCUT's semidefinite program [22] starts with a rank-one feasible solution, and then proceeds iteratively to improve its quality. In each iteration a rank-one update is applied to the current solution, so the rank of the current solution is increased by at most one. Since the number of iterations could as large as $\tilde{O}(\epsilon^{-2}n)$, the resulting solution is likely to be full-rank. However, it follows from Pataki's results that no matter how big the rank of the current solution is, there always exists a solution of rank $O(\sqrt{n})$ that has the same quality. Therefore our space-efficient algorithm is basically the time-efficient algorithm augmented with a number of *rank-reduction* steps, which was first proposed by Pataki [29]. Specifically, whenever the rank of the current solution is one more than the bound given by Pataki, we replace the current solution matrix by a new matrix that has one less rank, and has (almost) the same quality. That way we keep the rank of the current solution low. during the execution of their algorithm. As a result we reduce the space required by the currently best approximation algorithm for MAXCUT down to $O(m + n^{1.5})$, where $O(m)$ comes from the input.

COLORING. Our previous time-efficient algorithm for COLORING's semidefinite program [22] runs in $\tilde{O}(\epsilon^{-2})$ iterations. In each iteration an approximate solution for a MAXCUT semidefinite program is obtained. The resulting approximation solution of the COLORING semidefinite program is a linear combination of the solutions obtained from all iterations. If we resort to our space-efficient algorithm for MAXCUT semidefinite program in each of those iterations, then we are guaranteed to find an approximate solution for COLORING semidefinite program whose rank is $\tilde{O}(\epsilon^{-2}\sqrt{n})$. This gives an approximation algorithm for COLORING's semidefinite relaxation in space $O(m)+\tilde{O}(\epsilon^{-2}n^{1.5})$. It is interesting to observe that Pataki's results only guarantee a rank-$O(\sqrt{m})$ solution for the COLORING semidefinite program. Therefore the rank of the solution output by our algorithm is asymptotically less than the bound given by Pataki when ϵ is a constant and the graph is dense.

MAXCLIQUE. The only known polynomial-time algorithm for MAXCLIQUE on perfect graphs is based on the observation that the clique number for a perfect graph can be computed in polynomial time [13]. It follows from a result in the journal version of [19] that the clique number for a perfect graph is strongly related to the optimal value of its semidefinite relaxation of COLORING. Therefore we can use our space-efficient algorithm for COLORING's semidefinite relaxation to reduce the space requirement of the only known polynomial-time algorithm for finding a maximum clique on a perfect graph.

Outline. Here is the structure for the rest of the paper. We first give some preliminaries in the following two sections. In §5 we explain the rank-reduction procedure used by our space-efficient algorithms. It is interesting to note that our explanation can be regarded as an alternative proof for Pataki's rank bound on the basic solutions of semidefinite programs. In §6, §7, and §8 we give the space-efficient algorithm for MAXCUT's semidefinite program. In §9 we show the space-efficient algorithm for COLORING's semidefinite program. In §10 we show how to reduce the space required by the polynomial-time algorithm for solving MAXCLIQUE on perfect graphs.

3 Preliminaries

All matrices and vectors are real in the paper. All vectors are column vectors in the paper. Let x be an n-element vector. Define $\|x\|_2 = \sqrt{\sum_{1\le i\le n} x_i^2}$. Let A and B be two $m \times n$ matrices. Define $\|A\|_F = \sqrt{\sum_{1\le i\le m}\sum_{1\le j\le n} A_{ij}^2}$. $A \bullet B = \sum_{1\le i\le m}\sum_{1\le j\le n} A_{ij}B_{ij}$. We use $X \succeq 0$ to signify that X is symmetric and positive semidefinite. Vectors $v_1,\ldots,v_n$ are *orthonormal* if the following holds for every $1 \le i,j \le n$.

$$v_i^T v_j = \begin{cases} 1 \text{ when } i=j \\ 0 \text{ when } i \ne j. \end{cases}$$

Clearly if the columns of an $n \times k$ matrix U are orthonormal, then U^TU is the $k \times k$ identity matrix.

It is well-known that every $n \times n$ symmetric rank-k matrix X can be written as $X = VDV^T$ for some $k \times k$ diagonal matrix D and $n \times k$ matrix V, where the columns of V are orthonormal (see e.g. Theorem 2.5.4 of [17]). A few lemmas are required.

Lemma 1. *Let X be a symmetric matrix. Suppose $X = UDU^T$, where the columns of U are linearly independent, and D is $k \times k$ and diagonal.*

– *If the rank of X is k, then $D_{ii} \neq 0$ for every $1 \leq i \leq k$.*
– *$X \succeq 0$ if and only if $D_{ii} \geq 0$ for every $1 \leq i \leq k$.*

Lemma 2. *Let S and $\hat{S}$ be two symmetric $k \times k$ matrices, where $S \succeq 0$ and $\hat{S} \not\succeq 0$. Then $\hat{\xi} = \max\{\xi : S + \hat{\xi}\hat{S} \succeq 0\}$ is well-defined. Moreover the rank of $S + \hat{\xi}\hat{S}$ is at most $k - 1$.*

Lemma 3. *Let X, U, and S be three nonzero matrices such that $X = USU^T$. The following statements hold.*

– *The rank of X is no more than the rank of S.*
– *X is positive semidefinite if S is positive semidefinite.*

4 Compact Representation of Low-Rank Matrices

We need a procedure for the following problem: "Given an $n \times n$ symmetric matrix X of rank k, where k is unknown, but $k \leq \ell$ for some known $\ell \leq n$, compute an $n \times k$ matrix U and a symmetric $k \times k$ matrix S such that $X = USU^T$." Since the rank of X is k, there exists a $k \times k$ diagonal matrix D and an $n \times k$ matrix V, whose columns are orthonormal, such that $X = VDV^T$. Let v_i be the i^{th} column of V. Namely $X = \sum_{1 \leq i \leq k} \lambda_i v_i v_i^T$. In order to solve the above problem, we first compute ℓ vectors $w_1, \ldots, w_\ell$, where each w_i is obtained by multiplying X by a random vector r_i. Namely $w_i = Xr_i$, for every $1 \leq i \leq \ell$.

Lemma 4. *The linear space spanned by $v_1, \ldots, v_k$ is equal to the linear space spanned by $w_1, \ldots, w_\ell$ with probability one.*

Now we orthonormalize $w_1, \ldots, w_\ell$ using the Gram-Shmidt procedure (see e.g. Chapter 5 of [16]), and get a set of orthonormal vectors $u_1, \ldots, u_k$ such that the linear space spanned by $w_1, \ldots, w_\ell$ is equal to the linear space spanned by $u_1, \ldots, u_k$. Let U be the $n \times k$ matrix whose columns are $u_1, \ldots, u_k$. Since the linear space spanned by $u_1, \ldots, u_k$ is equal to the linear space spanned by $v_1, \ldots, v_k$, we know there exists a $k \times k$ matrix R such that $V = UR$. It follows that $X = URDR^TU^T$. Therefore we know there exists a symmetric $k \times k$ matrix $S = RDR^T$ such that $X = USU^T$. In fact $S = U^TXU$, since the columns of U are orthonormal. We then have a USU^T factoring for the given matrix X.

If the given matrix X is in the form of $\bar{U}\bar{S}\bar{U}^T$, where $\bar{U}$ is $n \times \bar{k}$ and $\bar{S}$ is $\bar{k} \times \bar{k}$, then the rank of X is at most $\bar{k}$ by Lemma 3. One can easily verify from the above that the USU^T decomposition of X can be found in time $O(n\bar{k}^2)$ and space $O(n\bar{k})$.

5 Bounding the Rank of Basic Solutions

Consider the following set of symmetric positive-semidefinite matrices $Q = \{X \succeq 0 : A^{(i)} \bullet X = b_i,\ i = 1, \ldots, m\}$, where X and $A^{(1)}, \ldots, A^{(m)}$ are $n \times n$

matrices. Pataki [28] shows that if Q nonempty, then Q contains a matrix of k, for some k such that $k(k+1) \leq 2m$. The low-rank matrix is called a *basic solution*, which is the analogy of a basic solution for a linear program [8]. His proof relies on the facial structure of the positive-semidefinite cone, which requires background from convex analysis [32,5]. Based on the existence of basic solutions, Pataki [29] also shows how to obtain a basic solution from a feasible solution by performing a sequence of rank-reduction procedure.

In the section we explain Pataki's rank-reduction procedure. Our explanation can be regarded as an alternative proof for Pataki's rank bound on basic matrix solutions. The key for the rank-reduction procedure is the following straightforward observation: "A homogeneous linear system has a nonzero solution if the number of variables is more than the number of constraints."

Pataki's bound on the rank of a basic solution follows inductively from the following lemma.

Lemma 5. *Suppose Q contains a matrix X whose rank is k, where $k(k+1) \geq 2m+1$. Then Q contains a matrix X' of rank at most $k-1$.*

6 The Algorithm for VECTOR MAXCUT

Let G be a graph composed of n nodes and m edges with positive weights. Let C be the matrix such that C_{ij} is the weight of edge ij. In [22] we give an algorithm VectorMaxCut(C, ϵ) for solving the following semidefinite program to within relative error ϵ.

$$\min\{\lambda : L \bullet X = 1; X \succeq 0; X_{ii} \leq \lambda, \text{for every } 1 \leq i \leq n\} \tag{1}$$

where L is the perturbed Laplacian matrix for the graph. At the end of the algorithm, the Cholesky decomposition of an ϵ-optimal solution is reported. The space required by VectorMaxCut is $\tilde{O}(n^2)$. In this section we show how to modify that algorithm and obtain an algorithm solving the same problem whose space requirement is only $O(m + n^{1.5})$, where $O(m)$ comes from the input.

CompactVectorMaxCut(C, ϵ), the compact version of the approximation algorithm, is as follows, where the procedure Initial(C) is given in [22].

1. Scale C such that the sum of edge weights is one, and then compute L from C. Let $(X, \lambda) = $ Initial(C). Let $\epsilon' = 1$.
2. While $\epsilon' > \epsilon$ do (a) Let $\epsilon' = \epsilon'/2$. (b) Let $(X, \lambda) =$ CompactImprove$(X, \lambda, \epsilon'/7)$.

The algorithm starts with finding an initial rank-one matrix (X, λ). It then iteratively calls CompactImprove to improve the quality of the current solution. The only thing that CompactVectorMaxCut differs from the original version VectorMaxCut is that the new subroutine CompactImprove runs in space $O(n^{1.5})$ and always outputs the matrix X in the form of USU^T, where U is $n \times k$ and S is $k \times k$, for some $k = O(\sqrt{n})$.

Define $\langle X \rangle_y = y_1 X_{11} + \cdots + y_n X_{nn}$. We give CompactImprove$(X, \lambda_0, \epsilon)$ as follows.

```
1. Let $\lambda = \lambda_0$. Let $\alpha = 12\epsilon^{-1}\ln(2n\epsilon^{-1})$. Let $\sigma = \epsilon/(4\alpha n)$.
2. Repeat
   (a) Let $y_i = e^{\alpha X_{ii}}$ for every $i = 1, \ldots, n$.
   (b) Let $\tilde{X} = \text{Direction}(y, \epsilon)$.
   (c) If $\langle X\rangle_y - \langle\tilde{X}\rangle_y \leq \epsilon(\langle X\rangle_y + \lambda y \cdot \mathbf{1})$    Return $(X, \lambda)$.
       else
         $X = (1-\sigma)X + \sigma\tilde{X}$. $X = \text{RankReduction}(X)$. $\lambda = \max\{X_{11}, \ldots, X_{nn}\}$.
```

Direction is given in [22], which runs in time $\tilde{O}(m\epsilon^{-1})$ and space $O(n)$. It always outputs a rank-one matrix in the form of $\tilde{u}\tilde{u}^T$ for some vector $\tilde{u}$.

The only thing that CompactImprove differs from the original version is that a new procedure RankReduction is called in each iteration when the rank-one update is applied to the current matrix. The purpose is to ensure that the rank k of the current matrix X satisfies that $k(k+1) \leq 2n+2$. Specifically, RankReduction$(\bar{X})$ computes the USU^T decomposition of $\bar{X}$ using the procedure described in §4, and therefore determines the rank k of $\bar{X}$. If $k(k+1) \geq 2n+3$, then it uses Pataki's rank-reduction procedure described in the proof of Lemma 5 to find a matrix $X' \in Q_{\bar{X}}$ whose rank is at most $k-1$, where $Q_{\bar{X}}$ is the set of matrices $\{X \succeq 0 : L \bullet X = L \bullet \bar{X},\ X_{ii} = \bar{X}_{ii},\ 1 \leq i \leq n\}$. Since $\bar{X} \in Q_{\bar{X}}$, we know $Q_{\bar{X}}$ is not empty. Lemma 5 guarantees the existence of X', which is also the output of RankReduction$(\bar{X})$.

In order to show that CompactVectorMaxCut is a space-efficient algorithm, it remains to show how to implement RankReduction to run in space $O(m + n^{1.5})$.

7 The Rank-Reduction Step

RankReduction$(\bar{X})$ has the following two steps.

1. Use the procedure given in §4 to compute the USU^T decomposition of $\bar{X}$ where the columns of U are orthonormal, and S is a $k \times k$ full-rank matrix.
2. If $k(k+1) \leq 2n+2$, then return $\bar{X}$ in the form of USU^T. Otherwise,
   (a) Find a symmetric $k \times k$ matrix $\hat{S}$ such that $\hat{S} \not\succeq 0$, $L \bullet (U\hat{S}U^T) = 0$ and the diagonal elements of $U\hat{S}U^T$ are all zero.
   (b) Find a nonsingular matrix R and two diagonal matrices D and $\hat{D}$ such that $S = RDR^T$ and $\hat{S} = R\hat{D}R^T$.
   (c) Compute $\hat{\xi} = \min\{-D_{ii}/\hat{D}_{ii} : \hat{D}_{ii} < 0\}$. Let $S' = S + \hat{\xi}\hat{S}$. Let $X' = US'U^T$.
   (d) Return X' in the form of $US'U^T$.

The input matrix $\bar{X}$ is $(1-\sigma)$ times the output of the previous iteration plus a rank-one update $\sigma\tilde{u}\tilde{u}^T$. Therefore is can be put in the form $\bar{U}\bar{S}\bar{U}^T$, where U is $n \times \bar{k}$ and $\bar{S}$ is $\bar{k} \times \bar{k}$ for some $\bar{k} = O(\sqrt{n})$. It follows from §4 that the first step can be done in time $O(n^2)$ and space $(n^{1.5})$.

Step 2-(b) can be done in time $O(k^3)$ and space $O(k^2)$ using, for example, Algorithm 8.7.1 in [16].

The real challenge comes from Step 2-(a), which requires to find a nonzero solution for the underconstrained homogeneous linear system $Ax = 0$ with $O(n)$ variables and $O(n)$ constraints. Note that $L \bullet (U\hat{S}U^T)$ is equal to $(U^T LU) \bullet \hat{S}$, where $U^T LU$ is a $k \times k$ obtainable in time $O(mk + nk^2)$. Also Note that the i^{th} diagonal element of $U\hat{S}U^T$ is exactly $u_i^T \hat{S} u_i$ where u_i is the i^{th} column of U^T. Therefore the corresponding matrix A can be represented sparsely in space $O(n^{1.5})$, although it is $O(n) \times O(n)$.

Direct methods such as Gaussian Elimination cannot be applied here to find a nonzero solution for $Ax = 0$, since it would take $O(n^2)$ space. Therefore we have to resort to iterative methods which can find an approximate solution in space $O(n)$. Since iterative methods only give us approximate solution, we have to ensure that the error does not affect the quality of the current solution matrix by too much.

When $k(k + 1) \geq 2n + 3$, ideally we want to find a rank-$(k - 1)$ matrix X' in $Q_{\bar{X}}$. Namely we want X' and $\bar{X}$ to have exactly the same potential, so the numbers of iteration required by CompactImprove and Improve will have the same bound as. Since the solution for the linear system has error, the corresponding X' will not be in $Q_{\bar{X}}$. However, it would be OK if the potential of X' is slightly more than that of $\bar{X}$.

Specifically by the analysis shown in [30] each iteration of the original Improve(X, λ_0, ϵ) given in [22] reduces the potential by a factor of $\Delta = \frac{\epsilon^2 \lambda^*}{4n}$. The number of iterations can therefore be shown to be $O(\epsilon^{-2} n \log(n\epsilon^{-1}))$. The number of iterations will be at most twice as many, even if each iteration reduces the potential only by a factor of $\Delta/2$. Clearly $(1 - \Delta)(1 + \Delta/2) \geq (1 - \Delta/2)$. Therefore it would be OK if the approximate solution for $Ax = 0$ gives a $\hat{S}$ such that adding $\hat{\xi}\hat{X}$ to $\bar{X}$ increases the potential of $\bar{X}$ by at most a factor of $\Delta/2$.

As defined in [22], the potential of a solution matrix X, where $L \bullet X = 1$, is $\sum_{1 \leq i \leq n} e^{\alpha X_{ii}}$. Let $\delta = \max_i |\hat{X}_{ii}|$. If we add $\hat{\xi}\hat{X}$ to the current matrix $\bar{X}$, then the increase of potential is at most a factor of $2\hat{\xi}\delta$, because $\sum_{1 \leq i \leq n} (e^{\alpha \bar{X}_{ii} + \hat{\xi}\delta} - e^{\alpha \bar{X}_{ii}}) = (e^{\hat{\xi}\delta} - 1) \sum_{1 \leq i \leq n} e^{\alpha \bar{X}_{ii}} \leq 2\hat{\xi}\delta \sum_{1 \leq i \leq n} e^{\alpha \bar{X}_{ii}}$, when $\hat{\xi}\delta$ is small. Therefore we can reduce the goal to be ensuring that $\hat{\xi}\delta \leq \frac{\Delta}{4}$. By the following lemma we know it suffices to guarantee that $\|\hat{X}\|_F \geq 1$ and $\delta \leq \frac{\Delta}{16n\bar{\lambda}}$, since $\frac{\Delta}{16n\bar{\lambda}}$ is clearly no more than $\frac{1}{2n^2}$.

Lemma 6. *If $\delta \leq \frac{1}{2n^2}$ and $\|\hat{X}\|_F \geq 1$ then $\hat{\xi} \leq 4n\bar{\lambda}$, where $\bar{\lambda}$ is the maximum diagonal element of $\bar{X}$.*

Therefore in each rank-reduction step, we find a nonzero solution $\hat{x}$ for a homogeneous linear system $Ax = 0$, and $\hat{S}$ can be obtained from $\hat{x}$. By the above argument we know that the norm of Ax, which is an upper bound of every $\hat{X}_{ii}$, is so small that the matrix $\hat{X} = U\hat{S}U^T$ has to satisfy (1). $\|\hat{X}\|_F \geq 1$, and (2) $|\hat{X}_{ii}| \leq \frac{\Delta}{16n\bar{\lambda}} = \frac{\epsilon^2 \lambda^*}{64n^2\bar{\lambda}} \leq \frac{\epsilon^2}{192n^2}$. One can verify that $\|A\|_F \leq 2n$. Therefore to find a sufficiently good $\hat{S}$, it suffices to find a nonzero vector x such that (1) $\|x\|_2 \geq 1$, and (2) $\|Ax\|_2 \leq \frac{\epsilon^2}{384n^3} \|A\|_F$.

8 Approximately Solving a Homogeneous Linear System in Exact Arithmetic

Given a homogeneous linear system $\bar{A}x = b$, where $\bar{A}$ is a nonsingular $n \times n$ matrix, and b is nonzero. Then an approximate solution to the system with error at most ϵ' can be found in time $O(n^3 \log(1/\epsilon'))$ in exact arithmetic and space $O(n)$ using iterative methods like the *conjugate gradient method* or the *steepest descent method* [34]. It can be shown that in our homogeneous system $Ax = 0$, the number of variables is $O(\sqrt{n})$ more than the number of constraints. Therefore we can throw in $O(\sqrt{n})$ linear constraints with random coefficients to make the system nonsingular and nonhomogeneous, and then use iterative methods to find an approximate solution of the new system, which also clearly gives a nonzero approximate solution for the original homogeneous system. Note that throw in those $O(\sqrt{n})$ constraints requires $O(n^{1.5})$ extra space, which fortunately does not exceed the space bound we are aiming for. As for exactly how many extra constraints have to be added to make the system nonsingular but still consistent, we can use binary search, which will add an $O(\log n)$ factor to the required running time.

Therefore we can achieve the goal shown at the end of the previous section in time $O(n^3 \log^2(n\epsilon^{-1}))$ and space $O(n^{1.5})$ using exact arithmetic. (In practice, however, the running time of iterative methods depends polynomially on the condition number of the matrix A, i.e. the largest ratio of the absolution values of two eigenvalues of A. Researchers have been developing various kinds of pre-conditioning techniques to bring down the condition number of the given linear system [6,21,33].)

Overall complexity. As shown in [22], the total number of iterations required by CompactVectorMaxCut(C, ϵ) is $O(\epsilon^{-2} n \log(n\epsilon^{-1}))$ and the running time of each iteration is dominated by the time required by solving the homogeneous linear system, which is $O(n^3 \log^2(n\epsilon^{-1}))$ using exact arithmetic. It follows that the time complexity of our space-efficient algorithm is $O(\epsilon^{-2} n^4 \log^3(n/\epsilon))$ using exact arithmetic. As shown in the previous sections, the space required is $O(m + n^{1.5})$.

9 The Algorithm for VECTOR COLORING

If the given graph is k-colorable, in [22] we show that an ϵ-optimal solution for COLORING's semidefinite relaxation can be found by making $O(\epsilon^{-1} \log(n\epsilon^{-1}))$ subroutine calls to VectorMaxCut, each of which finds an $\frac{\epsilon}{6k}$-optimal solution to a semidefinite relaxation for some graph MAXCUT. The resulting ϵ-optimal solution is simply a linear combination of the approximate solutions reported by those $\tilde{O}(\epsilon^{-2})$ calls to VectorMaxCut. Clearly we can replace VectorMaxCut by CompactVectorMaxCut. We therefore have an algorithm CompactVectorColoring(G, ϵ), which runs in space $O(m+n^{1.5}\epsilon^{-2}\log(n\epsilon^{-1}))$.

Since the matrix output by CompactVectorMaxCut has rank at most $\sqrt{2n}$, the rank of the solution output by CompactVectorColoring has rank $\tilde{O}(\epsilon^{-2}\sqrt{n})$. It is interesting to note that if ϵ is a constant, then the rank of the ϵ-optimal solution output by CompactVectorColoring is asymptotically lower than Pataki's rank bound $O(\sqrt{m})$ on the optimal solutions.

10 The Algorithm for MAXCLIQUE on Perfect Graphs

The only known polynomial-time algorithm for MAXCLIQUE on perfect graphs [13] is based on the observation that the clique number for a perfect graph can be computed in polynomial time [25,23]. In each iteration of the algorithm, the clique number of an induced subgraph, which is therefore perfect [26] and has less edges, has to be computed. Let G be a perfect graph. Let $\omega(G)$ be G's clique number. Let $\lambda(G)$ be the optimal value of G's semidefinite relaxation for COLORING. It is shown in the journal version of [19] that $\lambda(G) = 1/(1-\omega(G))$. If the given perfect graph is k-colorable, then it is well-known that $\omega(G) \leq k$. It follows that an $O(1/k)$-optimal approximation to $\lambda(G)$ can be used to determine $\omega(G)$, since $\omega(G)$ is an integer. It follows that our CompactVectorColoring gives a way to implement the algorithm given in [13] in space $O(m + k^2 n^{1.5} \log(nk))$.

References

1. F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995. 388
2. N. Alon and N. Kahale. Approximating the independence number via the θ-fucntion. Unpublished manuscript, Nov. 1994. 388
3. A. Blum and D. Karger. An $\tilde{O}(n^{3/14})$-coloring for 3-colorable graphs. Submitted to *Information Processing Letters*, Jan. 1996. 388
4. A. Blum, G. Konjevod, and R. Ravi. Semi-definite relaxations for minimum bandwidth and other vertex-ordering problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, 1998. 388
5. A. Brøndsted. *An Introduction to Convex Polytopes*. Springer-Verlag, New York Heidelberg Berlin, 1983. 392
6. A. M. Bruaset. *A Survey of Preconditioned Iterative Methods*. Longman Scientific & Technical, 1995. 395
7. B. Chor and M. Sudan. A geometric approach to betweenness. In *Proceedings of the Third Annual European Symposium Algorithms*, pages 227–237, 1995. 388
8. V. Chvátal. *Linear Programming*. Freeman, New York, 1983. 392
9. U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, 1995. 388
10. R. M. Freund. Complexity of an Algorithm for Finding an Approximate Solution of a Semi-Definite Program with no Regularity Assumption. Technical Report OR-302-94, Operations Research Center, M.I.T., 1994. 388
11. A. M. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, May 1997. 388
12. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, Nov. 1995. 387, 388

13. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. 387, 388, 390, 396
14. M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. *Annal of Discrete Mathematics*, 21:325–357, 1985. 387
15. C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2), May 1996. 388
16. G. H. Holub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, second edition, 1989. 391, 393
17. R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985. 390
18. F. Jarre. An interior-point method for minimizing the maximum eigenvalue of a linear combination of matrices. *SIAM Journal on Control and Optimization*, 31(5):1360–1377, 1993. 388
19. D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *35th FOCS*, pages 2–13. IEEE, 1994. 388, 390, 396
20. H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *38th Annual Symposium on Foundations of Computer Science*, pages 406–415, Miami Beach, Florida, 20–22 Oct. 1997. IEEE. 388
21. C. T. Kelly. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, 1995. 395
22. P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAXCUT and COLORING. In *28-th STOC*, pages 338–347, 1996. 388, 389, 392, 393, 394, 395
23. D. E. Knuth. The sandwich theorem. *The Electronic Journal of Combinatorics*, 1(A1):1–48, 1994. 396
24. M. Kojima, S. Shindoh, and S. Hara. Interior–Point Methods for the Monotone Linear Complementarity Problem in Symmetric Matrices. Technical Report B-282, Department of Information Sciences, Tokyo Inst. of Technology, 1994. 388
25. L. Lovász. On the Shannon Capacity of a graph. *IEEE Transactions on Information Theory*, IT-25:1–7, 1979. 387, 396
26. L. Lovász. Perfect graphs. In L. W. Beineke and R. J. Wilson, editors, *Selected Topics in Graph Theory 2*, pages 55–87. Academic Press, London, 1983. 396
27. Y. Nesterov and A. Nemirovskii. *Self-Concordant Functions and Polynomial Time Methods in Covex Programming.* Central Economic and Mathematical Institute, USSR Academy of Science, Moscow, 1989. 388
28. G. Pataki. On the facial structure of cone-LP's and semi-definite programs. Technical Report MSRR-595, Graduate School of Industrial Administration, Carnegie–Mellon University, 1994. 392
29. G. Pataki. Cone-LP's and semidefinite programs: Geometry and a simplex-type method. In *Proceedings of the Fifth IPCO Conference on Integer Programming and Combinatorial Optimization*, 1996. 388, 389, 392
30. S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *32nd FOCS*, pages 495–504. IEEE, 1991. 394
31. F. Rendl, R. Vanderbei, and H. Wolkowicz. A primal-dual interior-point method for the max-min eigenvalue problem. Technical report, University of Waterloo, Dept. of Combinatorics and Optimization, 1993. 388
32. R. T. Rockafellar. *Convex Analysis.* Princeton University Press, 1970. 392

33. Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996. 395
34. J. R. Shewchuk. An introduction to the Conjugate Gradient Method without the agonizing pain. Technical Report CMU-CS-94-125, Shool of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Mar. 1994. 395
35. L. Vandenberghe and S. Boyd. Semidefinite programming. Technical report, Information Systems Laboratory, Stanford University, 1994. 387
36. L. Vandenberghe and S. Boyd. A primal-dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming, Series B*, 69:205–236, 1995. 388
37. A. Yoshise. An optimization method for convex programs — interior-point method and analytical center. *Systems, Control and Information*, 38(3):155–160, Mar. 1994. 388

A Capacitated Vehicle Routing Problem on a Tree*

Shin-ya Hamaguchi[1] and Naoki Katoh[2]

[1] Information System Promotion Division, Osaka Prefectural Government
Osaka, 540-8570 Japan
HamaguchiS@mbox.pref.osaka.jp

[2] Department of Architecture and Architectural Systems, Kyoto University
Kyoto, 606-8501 Japan
naoki@archi.kyoto-u.ac.jp

Abstract. In this paper we deal with a vehicle routing problem on a tree-shaped network with a single depot. Customers are located on vertices of the tree, and each customer has a positive demand. Demands of customers are served by a fleet of identical vehicles with limited capacity. It is assumed that the demand of a customer is splittable, i.e., it can be served by more than one vehicle. The problem we are concerned with in this paper asks to find a set of tours of the vehicles with minimum total lengths. Each tour begins at the depot, visits a subset of the customers and returns to the depot without violating the capacity constraint. We show that the problem is NP-complete and propose a 1.5-approximation algorithm for the problem. We also give some computational results.

1 Introduction

In this paper we consider a capacitated vehicle routing problem on a tree-shaped network with a single depot. Let $T = (V, E)$ be a tree, where V is a set of n vertices and E is a set of edges, and $r \in V$ be a designated vertex called *depot*. Nonnegative weight $w(e)$ is associated with each edge $e \in E$, which represents the length of e. Customers are located at vertices of the tree, and a customer at $v \in V$ has a positive demand $D(v)$. Thus, when there is no customer at v, $D(v) = 0$ is assumed. Demands of customers are served by a set of identical vehicles with limited capacity. We assume throughout this paper that the capacity of every vehicle is equal to one, and that the demand of a customer is splittable, i.e., it can be served by more than one vehicle. Each vehicle starts at the depot, visits a subset of customers to (partially) serve their demands and returns to the depot without violating the capacity constraint. The problem we deal with in this paper asks to find a set of tours of vehicles with minimum total lengths to satisfy all the demands of customers. We call this problem TREE-CVRP.

Vehicle routing problems have long been studied by many researchers (see [5,6,7,11,12] for a survey), and are found in various applications such as

* Research of the second author is partly supported by the Grant-in-Aid of Ministry of Science, Culture and Education of Japan.

scheduling of truck routes to deliver goods from a warehouse to retailers, material handling systems and computer communication networks. Recently, AGVs (automated guided vehicle) and material handling robots are often used in manufacturing systems, but also in offices and hospitals, in order to reduce the material handling efforts. The tree-shaped network can be typically found in buildings with simple structures of corridors and in simple production lines of factories.

Vehicle scheduling problems on tree-shaped networks have recently been studied by several authors [3,8,9,15,16]. All of them treated in the literature deal with a single-vehicle scheduling that seeks to find an optimal tour under certain constraints other than those treated in this paper. To the authors' knowledge, TREE-CVRP has not been studied in the literature except [17] who considered the case where demand of each customer is not splittable and gave 2-approximation algorithm. For general undirected networks, the problem contains TSP (traveling salesman problem) as a special case, and thus it is not only NP-hard but APX-hard (except the result by [2] for Euclidean-CVRP in the plane). However, when restricted to tree-shaped networks, the complexity of TREE-CVRP is not clear. We shall show that TREE-CVRP is strongly NP-complete by a reduction from *bin-packing problem* (the proof is omitted from this extended abstract).

Thus, we turn our attention on developing approximate algorithms for the problem. Since TREE-CVRP is a special class of general CVRP, approximation algorithms originally developed for CVRP on general undirected networks can be used to approximately solve TREE-CVRP. In particular, the iterated tour partitioning (ITP) heuristic proposed by Haimovich and Rinooy Kan [13] and Altinkemer and Gavish [1] provides $1 + (1 - \frac{1}{k})\alpha$ approximation for such general CVRP when α-approximation algorithm for TSP is available, where the capacity of every vehicle is assumed to be equal to k and the demand of every customer is a positive integer. For instance, if the famous 1.5-approximate algorithm for TSP by Christofides [4] is used, the approximation factor becomes $2.5 - 1.5/k$. For tree-shaped networks, TSP can be optimally solved in a straightforward manner, and thus the direct consequence of [1,13] results in a $(2 - \frac{1}{k})$-approximation algorithm.

In this paper, we shall present an improved 1.5-approximation algorithm for TREE-CVRP by exploiting the tree structure of the network. The basic idea behind the algorithm is to (i) first find an appropriate subtree, to (ii) generate two routings to partially serve the demands of customers in the subtree by applying two different heuristics, and to (iii) choose the better one. We repeat this process until all the demands are served. We have implemented our algorithm and carried out computational experiments to see how effective our algorithm is. Compared with lower bounds, the results demonstrate that the solutions obtained by our algorithm are very close to optimal.

The rest of the paper is organized as follows. In Section 2, we give some necessary definitions and basic facts. Section 3 presents an algorithm and proves that its approximation ratio is 1.5. Section 4 presents an example for which the approximation factor becomes 1.25. Section 5 reports computational results. Section 6 concludes the paper.

2 Preliminaries

Since T is a tree, there exists a unique path between two vertices. For vertices $u, v \in V$, let $path(u, v)$ be the unique path between u and v. The length of $path(u, v)$ is denoted by $w(path(u, v))$. We often view T as a directed tree rooted at r. For a vertex $v \in V - \{r\}$, let $parent(v)$ denote the parent of v, and $C(v)$ the set of children of v. We assume throughout this paper that when we write an edge $e = (u, v)$, u is a parent of v unless otherwise stated. For any $v \in V$, let T_v denote the subtree rooted at v, and $w(T_v)$ and $D(T_v)$ denote the sum of weights of edges in T_v, and the sum of demands of customers in T_v, respectively. Since customers are located on vertices, customers are often identified with vertices.

Suppose that we are given a set $S \subset V - \{r\}$ with $\sum_{v\in S} D(v) \leq 1$. Then one vehicle is enough to serve all the demands of customers in S, and an optimal tour for S can be trivially obtained by first computing a minimal subtree T' that spans $S \cup \{r\}$ and by performing a depth-first search with r as the starting vertex. Thus, when we speak of a tour in this paper, we do not need explicitly give a sequence of vertices that a vehicle visits, but it is enough to specify a set of customers that the vehicle visits. Since the demand of a customer is splittable, in order to define a tour of a vehicle, we need to specify the amount of demand of each customer served by the vehicle.

A solution of TREE-CVRP consists of a set of tours. From the above discussion, we represent the tour of the j-th vehicle by

$$\{D_j(v) \mid v \in S_j\}, \tag{1}$$

where S_j is the set of customers for which some positive demands are served in the j-th tour, and $D_j(v)(> 0)$ for $v \in S_j$ is the amount of demand that the j-th vehicle serves at v. often referred to as the *optimal cost.*

For an edge $e = (u, v)$, let

$$LB(e) = 2w(e) \cdot \lceil D(T_v) \rceil. \tag{2}$$

$LB(e)$ represents a lower bound of the cost required for traversing edge e in an optimal solution because, due to the unit capacity of a vehicle, the number of vehicles required for any solution to serve the demands in T_v is at least $\lceil D(T_v) \rceil$ and each such vehicle passes e at least twice (one is in a forward direction and the other is in a backward direction). Thus, we have the following lemma.

Lemma 1. $\sum_{e\in E} LB(e)$ *gives a lower bound of the optimal cost of TREE-CVRP.*

3 Algorithm

A vertex $v \in V$ is called *D-feasible* if $D(T_v) \geq 1$ and is called *D-minimal* if it is *D-feasible* but none of its children are. The proposed algorithm first finds a *D-minimal* vertex, and determines a routing of one or two vehicles that partially serve demands of vertices in T_v by applying Strategy 1 or 2 depending on the cases described below. We then appropriately update the remaining demands of vertices visited by the routing currently determined. In addition, if the remaining demand of subtree T_v becomes zero, T_v as well as the edge $(parent(v), v)$ is deleted. In this section, we abuse the notations $D(v)$ and $D(T_v)$ to denote the remaining demands of vertex v or subtree T_v, respectively unless confusion occurs. We repeat this process until all the demands of T are served. Notice that, if there is no D-feasible vertex (i.e., $D(T_r) < 1$) any more, we can optimally serve the remaining demands by visiting the corresponding vertices in a depth-first manner.

The algorithm consists of a number of applications of Strategy 1 or 2. One application of Strategy 1 or 2 is called a *round*.

When a D-minimal vertex v is found, in order to determine the routing of one or two vehicles to partially serve the demands in T_v, we apply the following two strategies and choose the one with cheaper cost. Let $C(v) = \{v_1, v_2, \ldots, v_p\}$. We assume $D(T_{v_i}) > 0$ for every $v_i \in C(v)$ since otherwise such subtree can be eliminated from T. Let $S \subseteq C(v)$ satisfying $1 \leq \sum_{v_i \in S} D(T_{v_i}) < 2$. Since $D(T_{v_i}) < 1$ for all $v_i \in C(v)$ from D-minimality of v, such S always exists and can be easily computed as $\cup_{i=1}^{k} T_{v_i}$ satisfying

$$\sum_{i=1}^{k-1} D(T_{v_i}) < 1 \quad \text{and} \quad \sum_{i=1}^{k} D(T_{v_i}) \geq 1. \tag{3}$$

If $\sum_{i=1}^{k} D(T_{v_i}) = 1$, we simply allocate one vehicle to serve all the demands in $\cup_{i=1}^{k} T_{v_i}$. Thus, we assume otherwise. For the ease of the exposition of the algorithm, we assume $k = 2$ without loss of generality because otherwise we can equivalently modify T_v by creating a fictitious vertex v' of $D(v') = 0$ as well as edge (v, v') of zero weight and replacing edges (v, v_i) for i with $1 \leq i \leq k-1$ by (v', v_i) with $w(v', v_i) = w(v, v_i)$ and an edge (v, v') with zero weight.

With this assumption, the algorithm considers subtrees T_{v_1} and T_{v_2} satisfying

$$D(T_{v_1}) < 1, D(T_{v_2}) < 1 \quad \text{and} \quad D(T_{v_1}) + D(T_{v_2}) > 1. \tag{4}$$

The first strategy (Strategy 1) prepares two vehicles to serve all the demands in $T_{v_1} \cup T_{v_2}$, while the second strategy (Strategy 2) prepare one vehicle to partially serve the demands in $T_{v_1} \cup T_{v_2}$ by using its full capacity (thus, the demand of some vertex may possibly be split).

Strategy 1: We prepare one vehicle for T_{v_1} and another vehicle for T_{v_2} to separately serve demands of T_{v_1} and T_{v_2}. The cost to serve these demands is

$$4w(path(r, v)) + 2w(T_{v_1}) + 2w(T_{v_2}) \tag{5}$$

because two vehicles run on the path $path(r, v)$ but each of T_{v_1} and T_{v_2} is traversed by only one vehicle.

Strategy 2: We assume $w(T_{v_1}) \geq w(T_{v_2})$ without loss of generality. We split the demand $D(u)$ of every $u \in T_{v_2}$ into $D'(u)$ and $D''(u)$ so that

$$\sum_{u \in T_{v_1}} D(u) + \sum_{u \in T_{v_2}} D'(u) = 1. \tag{6}$$

We allocate one vehicle to serve the set of demands $\{D(u) \mid u \in T_{v_1}\}$ and $\{D'(u) \mid u \in T_{v_2}\}$. The computation of such $D'(u)$ satisfying (6) is straightforward, i.e., it is done by performing a depth-first search on T_{v_2} with v_2 as a starting vertex in such a way that

(1) Initially set $sum = \sum_{u \in T_{v_1}} D(u)$.

(2) Every time a new vertex u is visited, if $sum + D(u) \leq 1$ holds, we set $D'(u) = D(u)$, $D''(u) = 0$ and increment sum by $D(u)$, otherwise we set $D'(u) = 1 - sum$, $D''(u) = D(u) - D'(u)$ and increment sum by $D'(u)$.

Notice that the demand of at most one vertex is split by this procedure. The cost required for Strategy 2 is at most

$$2w(path(r, v)) + 2w(T_{v_1}) + 2w(T_{v_2}). \tag{7}$$

Demands of a subset of vertices in T_{v_2} remain unserved, and thus T_{v_2} (or its subgraph) may possibly be visited later by other vehicles more than once. Thus, in total the cost to visit T_{v_2} (or its subgraph) will be counted twice or more as (7). For the ease of the analysis of approximation ratio of the proposed algorithm, we amortize the the cost to visit T_{v_2} in the current round so that it is charged to T_{v_1}. Since $w(T_{v_1}) \geq w(T_{v_2})$, the cost of (7) is bounded from above by

$$2w(path(r, v)) + 4w(T_{v_1}). \tag{8}$$

We consider (8) as the cost for Strategy 2. An alternative interpretation of (8) is that a set of demands defined by $\{D'(u) \mid u \in T_{v_2}\}$ are served without visiting T_{v_2} (resp. T_{v_1}) at the expense of visiting T_{v_1} twice. Notice that the subtree T_{v_1} will never be visited in future rounds because all the demands therein are served in the current round.

It should be remarked that our algorithm chooses Strategy 1 or 2 not by directly comparing the costs of (5) and (8), but by the following rule.

Selection rule of Strategy 1 or 2: We apply Strategy 1 if

$$\frac{4w(path(r, v)) + 2w(T_{v_1}) + 2w(T_{v_2})}{2w(path(r, v)) + 2w(T_{v_1}) + 2w(T_{v_2})} \leq \frac{2w(path(r, v)) + 4w(T_{v_1})}{2w(path(r, v)) + 2w(T_{v_1})}, \tag{9}$$

and apply Strategy 2 otherwise.

The rationale behind this selection rule is as follows: Since the amounts of demands as well as the sets of customers served by Strategies 1 and 2 are different in general, it may not be fair to directly compare (5) and (8), but it is reasonable to compare the costs of (5) and (8) divided by the lower bounds of the costs to optimally execute their corresponding tasks. In fact, the denominators of the left-hand and right-hand sides of (9) stand for such lower bounds as will be seen below. Now we shall prove the following main theorem.

Theorem 1. *The approximation of our algorithm for TREE-CVRP is* 1.5.

Proof. The proof is done by induction on the number of rounds. When $D(T_r) \leq 1$, our algorithm trivially finds an optimal solution. This proves the base case.

Assuming that our algorithm computes a 1.5-approximate solution for problems that require at most k rounds, we prove that the theorem also holds for the case which requires $k+1$ rounds. Let P denote the problem instance of TREE-CVRP for which our algorithm requires $k+1$ rounds. Let us consider the first round for P in which a D-minimal subtree T_v is found. As we remarked earlier, let us assume without loss of generality that the algorithm considers two T_{v_1} and T_{v_2} satisfying (4) and that $w(T_{v_1}) \geq w(T_{v_2})$. Depending whether (9) holds or not, Strategy 1 or 2 is applied by which one or two tours are determined to serve the demands of customers in $T_{v_1} \cup T_{v_2}$. Let P' be the problem instance of TREE-CVRP obtained from P after the first round by decreasing demands served in this round from original $D(\cdot)$. Let d_{sum} denote the total amount of demands served in this round. For each vertex $u \in V$, let $D_{remain}(u)$ and $D_{remain}(T_u)$ denote the remaining demands for u and subtree T_u, respectively. Thus, for each edge $e = (x, u) \in path(r, v)$, we have

$$D_{remain}(T_u) = D(T_u) - d_{sum}. \tag{10}$$

Notice that $d_{sum} \geq 1$ holds. Let $d(u)$ denote the demand of $u \in T_{v_1} \cup T_{v_2}$ served by this round. We then have $D_{remain}(u) = D(u) - d(u)$ for $u \in T_{v_1} \cup T_{v_2}$. For vertices u that belongs to neither $path(r, v)$ or $T_{v_1} \cup T_{v_2}$, $D_{remain}(u) = D(u)$.

Let $LB(P')$ be the lower bound for the problem P'. From Lemma 1, we then have

$$LB(P') = 2\Big(\sum_{\substack{u \in path(r,v) \text{ or} \\ u \in T_{v_1} \cup T_{v_2}}} \lceil D_{remain}(T_u) \rceil w(e) + \sum_{\substack{u \notin path(r,v) \text{ and} \\ u \notin T_{v_1} \cup T_{v_2}}} \lceil D(T_u) \rceil w(e)\Big), \tag{11}$$

where e is the edge connecting u and its parent. Let $cost(P)$, $cost_1$ and $cost(P')$ denote the total cost required for the original problem P by our algorithm, the cost required by the first round and the cost for the remaining problem P' to be required by our algorithm, respectively, (i.e., $cost(P) = cost_1 + cost(P')$). Let $LB(P)$ denote the lower bound of the optimal cost for P given by Lemma 1.

(Case 1): (9) holds. Thus, the algorithm applies Strategy 1. Since $\lceil x \rceil \geq 1 + \lceil x - a \rceil$ holds in general for any positive x, a with $x \geq a$ and $a \geq 1$, it follows from (10) and $d_{sum} \geq 1$ that $\lceil D(T_u) \rceil \geq 1 + \lceil D_{remain}(T_u) \rceil$ holds for each $u \in path(r, v)$. Also, $LB(e) = 2w(e)$ holds for all $e \in T_{v_1} \cup T_{v_2}$ from $D(T_{v_1}) < 1$ and $D(T_{v_2}) < 1$, and $D_{remain}(u) = 0$ holds for all $u \in T_{v_1} \cup T_{v_2}$ from the way of Strategy 1. Thus, we have

$$LB(P) \geq 2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2}) + LB(P'). \tag{12}$$

From (5), we have

$$cost_1 = 4w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2}).$$

Thus,

$$\begin{aligned}\frac{cost(P)}{LB(P)} &\leq \frac{cost_1 + cost(P')}{2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2}) + LB(P')}\\ &= \frac{4w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2}) + cost(P')}{2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2}) + LB(P')}.\end{aligned} \tag{13}$$

Since $cost(P')/LB(P') \leq 1.5$ holds from the induction hypothesis, it suffices to prove

$$\frac{4w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})}{2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})} \leq 1.5. \tag{14}$$

Since

$$\min\{\frac{b}{a}, \frac{d}{c}\} \leq \frac{b+d}{a+c} \tag{15}$$

holds for any positive a, b, c, d, it follows from (9) that

$$\frac{4w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})}{2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})} \leq \frac{6w(path(r,v)) + 6w(T_{v_1}) + 2w(T_{v_2})}{4w(path(r,v)) + 4w(T_{v_1}) + 2w(T_{v_2})} \leq 1.5.$$

Thus, (14) holds, and hence $cost(P)/LB(P) \leq 1.5$ follows from (13).

(Case 2): (9) does not hold, i.e,

$$\frac{4w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})}{2w(path(r,v)) + 2w(T_{v_1}) + 2w(T_{v_2})} > \frac{2w(path(r,v)) + 4w(T_{v_1})}{2w(path(r,v)) + 2w(T_{v_1})}, \tag{16}$$

i.e., Strategy 2 is applied. In this case, $d_{sum} = 1$ holds since one vehicle is scheduled to serve demands in $T_{v_1} \cup T_{v_2}$ with its full capacity. Thus, from (10), $\lceil D(T_u) \rceil = 1 + \lceil D_{remain}(T_u) \rceil$ holds for each $u \in path(r,v)$. Since all the demands of T_{v_1} are served in this round from the way of Strategy 2, $D_{remain}(T_u) = 0$ holds for all $u \in T_{v_1}$. Thus, from (2) we have

$$LB(P) \geq 2w(path(r,v)) + 2w(T_{v_1}) + LB(P'). \tag{17}$$

From (8), we have

$$cost_1 = 2w(path(r,v)) + 4w(T_{v_1}).$$

Thus,

$$\frac{cost(P)}{LB(P)} \leq \frac{cost_1 + cost(P')}{2w(path(r,v)) + 2w(T_{v_1}) + LB(P')} = \frac{2w(path(r,v)) + 4w(T_{v_1}) + cost(P')}{2w(path(r,v)) + 2w(T_{v_1}) + LB(P')}. \tag{18}$$

Since $cost(P')/LB(P') \leq 1.5$ holds from the induction hypothesis, it suffices to prove

$$\frac{2w(path(r,v)) + 4w(T_{v_1})}{2w(path(r,v)) + 2w(T_{v_1})} \leq 1.5. \tag{19}$$

Using (15) again, we can prove (19) in a manner similar to Case 1 because it holds from (15) and (16) that

$$\frac{2w(path(r,v)) + 4w(T_{v_1})}{2w(path(r,v)) + 2w(T_{v_1})} \leq \frac{6w(path(r,v)) + 6w(T_{v_1}) + 2w(T_{v_2})}{4w(path(r,v)) + 4w(T_{v_1}) + 2w(T_{v_2})} \leq 1.5.$$

Thus, $cost(P)/LB(P) \leq 1.5$ follows.

Although the details are omitted, it is easy to see that the algorithm can be implemented in such a way that each round runs in linear time. Thus, the total running time of our approximate algorithm is $O(\sum_{v\in V} D(v)\cdot n)$ time since the number of rounds required by the algorithm is clearly $O(\sum_{v\in V} D(v))$. This is polynomial if $\sum_{v\in V} D(v)$ is polynomial in n.

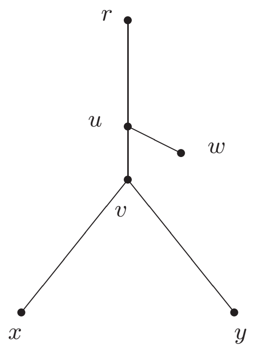
4 Lower Bound

We shall show that the approximation ratio of the proposed algorithm is at least 1.25 by giving such an example which is illustrated in Figure 1. In the figure, ϵ is a sufficiently small positive constant.

First of all, the algorithm finds a D-minimal vertex v and chooses Strategy 1 since (9) holds, and the cost of the first round required by the algorithm is 8. There still remains a demand of 0.6 at w, which requires cost of 2. Thus, the total cost is 10. On the other hand, the optimal schedule is that the first vehicle serves the demand of 0.6 at x and on the way back to the depot, visits vertex w to serve the demand of 0.3, which requires cost $4+2\epsilon$ and that the second serves the remaining demands with the same cost $4+2\epsilon$. Thus, the optimal cost is $8+4\epsilon$. Therefore, the approximation ratio for this example is 1.25.

5 Computational Results

We have implemented our algorithm and the ITP heuristic developed by [1,13] in order to see the practical performance of our algorithm by comparing with ITP heuristic. For this, we have generated 10 problem instances for each of $n = 50, 100, 150, 200, 250, 300$ according to the following scheme. For each n,



$$w(r, u) = 1 - \epsilon$$

$$w(u, v) = w(u, w) = \epsilon$$

$$w(v, y) = w(v, y) = 1$$

$$D(r) = D(u) = D(v) = 0$$

$$D(w) = D(x) = D(y) = 0.6$$

Fig. 1. Illustration of the worst case example

problem instances are randomly generated as follows: Starting with a single vertex corresponding to a depot, we randomly pick a leaf in the currently generated tree, and generate its children whose number is determined by uniform distribution over the integers in $[0, 4]$. The weights of edges are determined by uniform distribution over $[0, 20]$. The demands of vertices are also determined by uniform distribution over $[0, 1]$.

The performance of the algorithms are evaluated by comparing the ratios of costs of solutions produced by the algorithms to the lower bound given by Lemma 1. The overall average of such ratio of our algorithm is 1.016, while that of ITP is 1.113. The worst-case ratio among 60 instances is 1.072 for our algorithm while that for ITP heuristic is 1.238. For 37 cases out of 60 instances generated, our algorithm produced solutions that match the lower bound, while there was only one such case for ITP.

From this experiments, we can observe that our algorithm is superior to ITP heuristic, and that it practically produces solutions very close to optimal.

More extensive computational experiments are reported in [14] which conducted experiments by varying parameters in several ways to generate various types of problem instances and by practically refining the algorithm. However, the general tendency seen above remains unchanged.

Therefore, we can conclude that the proposed algorithm is practically effective as well.

6 Conclusion

We have developed an efficient 1.5-approximation algorithm for the capacitated vehicle routing problem on tree-shaped networks where demand of each customer is allowed to be split. (Notice that when the demand is not allowed to be split, using Strategy 1 alone trivially produces 2-approximation algorithm.) We also gave an example for which the approximation ratio of our algorithm is 1.25.

There are still several problems that remain unsolved yet. First of all, we believe that we can further improve the approximation ratio by refining the analysis or the algorithm. Another direction of future research is to establish an inapproximability result for the problem.

References

1. K. Altinkemer and B. Gavish, Heuristics for delivery problems with constant error guarantees, *Transportation Science*, 24 (1990), 294-297. 398, 404
2. T.Asano, N.Katoh, H.Tamaki and T.Tokuyama, Covering points in the plane by k-tours : towards a polynomial time approximation scheme for general k, *Proc. of 29th Annual ACM Symposium on Theory of Computing*, pp.275-283, 1997. 398
3. I. Averbakh and O. Berman, Sales-delivery man problems on treelike networks, *Networks*, 25 (1995), 45-58. 398
4. N. Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, Report 388, Graduate School of Industrial Administration, 1976. 398
5. N. Christofides, A. Mingozzi and P. Toth. The vehicle routing problem. in: N. Christofides, A. Mingozzi, P. Toth and C. Sandi, editors. *Combinatorial Optimization.* John Wiley & Sons Ltd, London,1979. 397
6. M. Desrochers, J. K. Lenstra and M. W. P. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *Eur. J. Oper. Res.* 46, 322–332, 1990. 397
7. M.L. Fischer. Vehicle Routing. in *Network Routing,* Handbooks in Operations Research and Management Science, **8**, Ball, M. O., T. L. Magnanti, C. L. Monma and G. L. Nemhauser (Eds.), Elsevier Science, Amsterdam, 1-33, 1995. 397
8. G. Frederickson, Notes on the complexity of a simple transportation problem, *SIAM J. Computing*, 22-1 (1993), 57-61. 398
9. G. Frederickson and D. Guan, Preemptive ensemble motion planning on a tree, *SIAM J. Computing*, 21-6 (1992), 1130-1152. 398
10. M.R.Garey , D.S.Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
11. B.L. Golden. Introduction to and Recent Advances in Vehicle Routing Methods. in *Transportation Planning Models,* M. Florian (Ed), Elsevier Sci. Publ. B. V. (North Holland), 383-418, 1984. 397
12. B.L. Golden and A. A. Assad (Eds.). *Vehicle Routing: Methods and Studies,* Studies in Manag. Science and Systems 16, North-Holland Publ., Amsterdam, 1988. 397
13. M. Haimovich and A.H.G. Rinnooy Kan. Bounds and Heuristics for capacitated routing problems *Mathematics of Operations Research*, **10**(4), 527-542, 1985. 398, 404
14. S. Hamaguchi, A study on Vehicle Routing Problems on Tree-shaped Networks, Master Thesis, Graduate School of Business Administration, Kobe Univ. of Commerce, 1998. 405
15. Y. Karuno, H. Nagamochi, T. Ibaraki, Vehicle Scheduling on a Tree with Release and Handling Times, Proc. of ISAAC'93, *Lecture Notes in Computer Science 762, Springer-Verlag* 486-495, 1993. 398
16. Y. Karuno, H. Nagamochi, T. Ibaraki, Vehicle Scheduling on a Tree to Minimize Maximum Lateness, *Journal of the Operations Research Society of Japan*, Vol. 39, No.3 (1996) 345-355. 398

17. M. Labbé, G. Laporte and H. Mercure. Capacitated Vehicle Routing Problems on Trees, *Operations Research*, Vol. 39 No. 4 (1991) 616-622. 398
18. G.Laporte. The Vehicle Routing Problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59 (1992) 345-358.
19. J.K.Lenstra and A.H.G.Rinooy Kan. Complexity of Vehicle Routing and Scheduling Problem. *Networks*, 11 (1981) 221-227.
20. K. Lund VRP References - Part I/II. WWW home page, http : //www.imm.dtu.dk/documents/users/kl/vrp_1.html

Approximation Algorithms for Some Optimum Communication Spanning Tree Problems

Bang Ye Wu[1], Kun–Mao Chao[2], and Chuan Yi Tang[1]

[1] Dept. of Computer Science, National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
{dr838305,cytang}@cs.nthu.edu.tw

[2] Dept. of Computer Science and Information Management, Providence University
Shalu, Taiwan, R.O.C.
kmchao@csim.pu.edu.tw

Abstract. Let $G = (V, E, w)$ be an undirected graph with nonnegative edge weight w, and r be a nonnegative vertex weight. The product-requirement optimum communication spanning tree (PROCT) problem is to find a spanning tree T minimizing $\sum_{i,j \in V} r(i)r(j)d(T, i, j)$, where $d(T, i, j)$ is the distance between i and j on T. The sum-requirement optimum communication spanning tree (SROCT) problem is to minimize $\sum_{i,j \in V}(r(i) + r(j))d(T, i, j)$. Both the two problems are special cases of the general optimum communication spanning tree problem, and are generalizations of the shortest total path length spanning tree problem. In this paper, we present an $O(n^5)$ time 1.577-approximation algorithm for the PROCT problem, and an $O(n^3)$ time 2-approximation algorithm for the SROCT problem, where n is the number of vertices.

Keywords: approximation algorithms, spanning trees, network design.

1 Introduction

Consider the following network design problem: $G = (V, E, w)$ be an undirected graph with nonnegative edge weight w. The vertices may represent cities and the edge weights represent the distances. We are also given the requirements $l_{i,j}$ for each pair of vertices, which may represent the number of telephone calls between the two cities. For any spanning tree T of G, the routing cost between two cities is defined to be the requirement multiplied by the path length of the two cities on T, and the routing cost of T is total routing cost summed over all pairs of vertices. Our goal is to construct a spanning tree with minimum routing cost. That is, we want to find a spanning tree T such that $\sum_{i,j \in V} l_{i,j} d(T, i, j)$ is minimum, where $d(T, i, j)$ is the distance between i and j on T.

The above problem is called the *optimum communication spanning tree* (OCT) problem [3]. When the requirements are all equal to one, the problem is reduced to the *minimum routing cost spanning tree* (MRT) problem (or called as shortest total path length spanning tree problem) and was shown to be NP-hard in [4] (also listed in [2]). A 2-approximation algorithm was presented in [5]. Recently, a *polynomial time approximation scheme* (PTAS) for the MRT problem was presented in [6].

In this paper, we consider two generalizations of the MRT problem, which are also special cases of the OCT problem. Given a vertex weight r, we first consider a special case of the OCT problem in which the requirement $l_{i,j} = r(i) \times r(j)$. The vertex weight may represent the population of the city, and the communication requirement is assumed proportional to the product of the population of the two cities. We call such a problem the *product-requirement OCT* (PROCT). For the PROCT problem, we present a 1.577-approximation algorithm with time complexity $O(n^5)$. For an input graph $G = (V, E, w)$ of a PROCT problem, our algorithm first construct its metric closure, a complete graph in which edge weights are the shortest path lengths on G. The algorithm then find the minimum 2-star on the metric closure. A 2-star is a spanning tree with at most 2 internal nodes, and the minimum 2-star is the 2-star with minimum routing cost. Finally, it transfers the 2-star back to a spanning tree on G. To ensure the performance ratio, we construct a 2-star from the optimal solution and show that such a 2-star is a 1.577-approximation solution. We also show that the minimum 2-star can be found in $O(n^5)$ time by solving $O(n^2)$ minimum cut problems.

The second problem considered in this paper is the *sum-requirement optimum communication spanning tree* (SROCT) problem. Given a vertex weight r, the communication requirement between two nodes is defined to be $r(i) + r(j)$. The SROCT problem may arise from the following scenario: For each node in the network, there is an individual message to be sent to every other node and the amount of the message is propotional to the weight of the receiver. The goal is to find a spanning tree minimizing the total routing cost of the system. We show that the SROCT problem can be approximated within error ratio 2 and with time complexity $O(n^3)$. Our work is to show that there exists a vertex v such that the *shortest path tree* rooted at v is a 2-approximation solution.

The remaining sections are organized as follows: In Section 2, some definitions and notations are given. The PROCT problem is discussed in Section 3, and the SROCT problem is presented in Section 4.

2 Definitions and Notations

In this paper, a graph $G = (V, E, w)$ is a simple, connected, undirected graph, in which w is the nonnegative edge weight, not necessarily a metric. For any graph G, $V(G)$ denotes its vertex set and $E(G)$ denotes its edge set.

Definition 1. A metric graph $G = (V, V \times V, w)$ is a complete graph in which w is a nonnegative edge weight and satisfies the triangle inequality, that is, $w(i, j) + w(j, k) \geq w(i, k)\ \forall i, j, k \in V$.

Definition 2. Let $G = (V, E, w)$ be a graph and r be a vertex weight. $w(G) = \sum_{e \in E} w(e)$ and $r(U) = \sum_{i \in U} r(i)$, $\forall U \subset V(G)$. For $i, j \in V$, $SP(G, i, j)$ denotes a shortest path between i,j on G. $d(G, i, j) = w(SP(G, i, j))$.

Definition 3. Let T be a rooted tree. For any $v \in V(T)$, T_v denotes the subtree with root v.

Definition 4. Given a graph $G = (V, E, w)$ and a nonnegative requirement l over all pairs of vertices, the optimum communication spanning tree (OCT) problem is to find a spanning tree T such that $\sum_{i,j \in V} l_{i,j} d(T, i, j)$ is minimum among all possible spanning trees.

Definition 5. The minimum routing cost spanning tree (MRT) problem has the same definition as the OCT problem except that $l_{i,j} = 1\ \forall i, j \in V$. That is , we want to minimize $C(T) = \sum_{i,j \in V} d(T, i, j)$. The product-requirement OCT (PROCT) problem has the same definition except that there exists a nonnegative vertex weight r such that $l_{i,j} = r(i)r(j)\ \forall i, j \in V$. That is, we want to minimize $C_p(T) = \sum_{i,j \in V} r(i)r(j)d(T, i, j)$. The ΔPROCT problem is the PROCT problem whose input is a metric graph. The sum-requirement OCT (SROCT) problem has the same definition except that $l_{i,j} = r(i) + r(j)\ \forall i, j \in V$. That is, we want to minimize $C_s(T) = \sum_{i,j \in V} (r(i) + r(j))d(T, i, j)$.

Definition 6. The *metric closure* of G is the complete weighted graph $\bar{G} = (V(G), V(G) \times V(G), \delta)$, where $\delta(i, j) = d(G, i, j)\ \forall i, j \in V(G)$. Note that δ is a metric.

3 The Product-Requirement OCT Problem

In this section, we discuss the PROCT problem. Let $G = (V, E, w)$ and vertex weight r be the input of the PROCT problem. Our algorithm works as follows:

- Construct $\bar{G}$ from G.
- Find the minimum 2-star T of $\bar{G}$.
- Transfer T into a spanning tree Y of G with $C_p(Y) \le C_p(T)$.

The result of the PROCT problem is listed in the following theorem:

Theorem 1. There is a 1.577-approximation algorithm with time complexity $O(n^5)$ for the PROCT problem.

For the correctness of the theorem, we show the following in next subsections:

- Given any spanning tree T of $\bar{G}$, the transformation algorithm can transfer T into a spanning tree Y of G with $C_p(Y) \le C_p(T)$.
- For any $\varepsilon > 0$, if T is a (1+ε)-approximation solution of the ΔPROCT problem with input $\bar{G}$, then Y is a (1+ε)-approximation solution of the PROCT problem with input G.
- The minimum 2-star T is a 1.577-approximation solution of the ΔPROCT problem with input $\bar{G}$.
- The overall time complexity is $O(n^5)$.

3.1 A Reduction from the General to the Metric Case

In this subsection, we discuss the transformation algorithm and the related results. The algorithm comes from [6]. It was developed for the MRT problem, and we show that it also works for the PROCT problem. Let $G = (V, E, w)$ and $\bar{G} = (V, V \times V, \delta)$. Any edge (a, b) in $\bar{G}$ is called a *bad edge* if $(a, b) \notin E$ or $w(a, b) > \delta(a, b)$. Given any spanning tree T of $\bar{G}$, the algorithm first computes $SP(G, i, j)\ \forall i, j \in V$. Y is constructed by iteratively replacing the bad edges until there is no any bad edge. Since Y has no bad edge, $\delta(e) = w(e)\ \forall e \in E(Y)$, and Y can be thought as a spanning tree of G with the same cost. The algorithm is listed below.

```
**Algorithm** Remove_bad
**Input**: a spanning tree $T$ of $\bar{G}$
**Output** : a spanning tree $Y$ of $G$ such that $C_p(Y) \le C_p(T)$.
Compute all-pairs shortest paths of $G$.
**while** there exists a bad edge in $T$
    Pick a bad edge $(a, b)$. Root $T$ at $a$.
    /* assume $SP(G, a, b) = (a, x, ..., b)$ and $y$ is the father of $x$ */
    **if** $b$ is not an ancestor of $x$ **then**
        $Y^* = T \cup (x, b) - (a, b); Y^{**} = Y^* \cup (a, x) - (x, y);$
    **else**
        $Y^* = T \cup (a, x) - (a, b); Y^{**} = Y^* \cup (b, x) - (x, y);$
    **endif**
    **if** $C_p(Y^*) < C_p(Y^{**})$ **then** $Y = Y^*$ **else** $Y = Y^{**}$ **endif**
    $T = Y$
**endwhile**
```

The result is in the following lemma.

Lemma 2. Given a spanning tree T of $\bar{G}$, the algorithm Remove_bad constructs a spanning tree Y of G with $C_p(Y) \le C_p(T)$ in $O(n^3)$ time.

Lemma 2 can be proved by the following two claims. The next claim was proved in [6], and we omit it here.

Claim 3: The loop is executed at most $O(n^2)$ times.

Claim 4: In each iteration, $\min\{C_p(Y^*), C_p(Y^{**})\} \le C_p(T)$.

Proof. (outline) For any node v, let $S_v = V(T_v)$.
Case 1: $x \in S_a - S_b$. If $C_p(Y^*) \le C_p(T)$, the result follows. Otherwise, let $U_1 = S_a - S_b$ and $U_2 = S_a - S_b - S_x$. Since the distance does not change for any two vertices both in U_1 (or both in S_b), we have

$$C_p(T) < C_p(Y^*)$$
$$\sum_{i \in U_1} \sum_{j \in S_b} r(i) r(j) d(T, i, j) < \sum_{i \in U_1} \sum_{j \in S_b} r(i) r(j) d(Y^*, i, j)$$

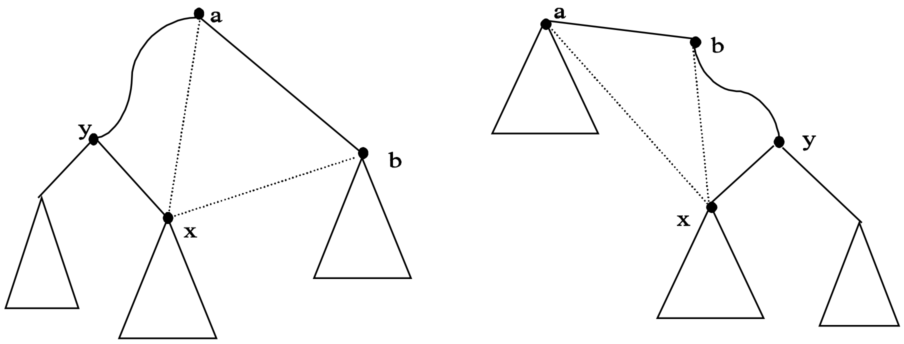


Fig. 1. Remove bad edge (a, b). Case 1 (left) and Case 2 (right).

Using $d(T, i, j) = d(T, i, a) + \delta(a, b) + d(T, b, j)$ and $d(Y^*, i, j) = d(T, i, x) + \delta(x, b) + d(T, b, j)$ $\forall i \in U_1, j \in S_b$, we have

$$\sum_{i \in U_1} r(i) \left(d(T, i, a) - d(T, i, x) \right) < -r(U_1)\delta(a, x)$$

Then since $d(Y^{**}, i, j) \leq d(T, i, j)$ for $i \in U_1$ and $j \in S_b$,

$$\left(C_p(Y^{**}) - C_p(T) \right)/2 \leq \sum_{i \in U_2} \sum_{j \in S_x} r(i)r(j) \left(d(Y^{**}, i, j) - d(T, i, j) \right)$$

Since $d(T, i, j) = d(T, i, x) + d(T, x, j)$ and $d(Y^{**}, i, j) = d(T, i, a) + \delta(a, x) + d(T, x, j)$ $\forall i \in U_2, j \in S_x$, we have

$$\left(C_p(Y^{**}) - C_p(T) \right)/2 \leq \sum_{i \in U_2} \sum_{j \in S_x} r(i)r(j) \left(d(T, i, a) + \delta(a, x) - d(T, i, x) \right)$$

$$\leq r(S_x) \sum_{i \in U_1} r(i) \left(d(T, i, a) - d(T, i, x) \right) + r(U_2)r(S_x)\delta(a, x)$$

$$< -r(U_1)r(S_x)\delta(a, x) + r(U_2)r(S_x)\delta(a, x) \leq 0$$

So, $C_p(Y^{**}) < C_p(T)$
Case 2: $x \in S_b$. Simiar as Case 1 and omitted. □

Let $PROCT(G)$ denote the optimum tree of the PROCT problem with input graph G. From Lemma 2, $C_p(PROCT(G)) \leq C_p(PROCT(\bar{G}))$. Since $d(G, u, v) \leq w(u, v)$ $\forall (u, v) \in E(G)$, $C_p(PROCT(G)) \geq C_p(PROCT(\bar{G}))$. We have the following corollaries.

Corollary 5. $C_p(PROCT(G)) = C_p(PROCT(\bar{G}))$.

Corollary 6. If there is a $(1+\varepsilon)$-approximation algorithm for ΔPROCT problem with time complexity $O(f(n))$, then there is a $(1+\varepsilon)$-approximation algorithm for PROCT with time complexity $O(f(n) + n^3)$.

3.2 An Algorithm for Minimum 2-Star

In this subsection, we present an algorithm for finding the 2-star T in which $C_p(T)$ is minimum among all possible 2-stars. We define the notation for a 2-star as follows:

Definition 7. Let $G = (V, E, w)$ be a metric graph. A 2-star of G is a spanning tree of G with at most two internal nodes. Assume $x \in X$ and $y \in Y$, and X,Y be a partition of V. We use $2star(x, y, X, Y)$ to denote a 2-star with edge set $\{(x, v)|\forall v \in X\} \cup \{(y, v)|\forall v \in Y\} \cup \{(x, y)\}$.

By definition, the following lemma is trivial, and we omit the proof.

Lemma 7. If $T = 2star(x, y, X, Y)$, and $R = \sum_{v \in X \cup Y} r(v)$

$$C_p(T) = 2r(X)r(Y)w(x,y) \\ +2\sum_{v\in X}(R - r(v))w(x,v) + 2\sum_{v\in Y}(R - r(v))w(y,v)$$

For any specified x and y, if we can find the best partition X and Y in $O(f(n))$ time, we can solve the minimum 2-star problem in $O(n^2 f(n))$ time by trying all possible vertex pairs. To find the best partition, we construct an undirected complete graph $H = (V, V \times V, h)$. The edge weight h is as follows:

1. $h(x, y) = 2r(x)r(y)w(x, y)$
2. $h(x, v) = 2(R - r(v))w(y, v) + 2r(v)r(x)w(x, y)$, and
   $h(y, v) = 2(R - r(v))w(x, v) + 2r(v)r(y)w(x, y)$ $\forall v \notin \{x, y\}$
3. $h(u, v) = 2r(u)r(v)w(x, y)$ $\forall u, v \notin \{x, y\}$

Let V_1 and V_2 be two subsets of V. (V_1, V_2) is said to be an x-y cut of H if (V_1, V_2) forms a partition of V and $x \in V_1$ and $y \in V_2$. The cost of an x-y cut (V_1, V_2) is defined to be $cut(V_1, V_2) = \sum_{u \in V_1, v \in V_2} h(u, v)$. The following lemma comes directly from the above construction and the proof is omitted.

Lemma 8. $cut(V_1, V_2) = C_p(2star(x, y, V_1, V_2))$.

The above lemma implies that we can find the minimum 2-star by solving $O(n^2)$ minimum cut problems. Since the minimum cut of a graph can be found in $O(n^3)$ (e.g. [1]), we have the following lemma:

Lemma 9. The minimum 2-star can be found in $O(n^5)$ time.

3.3 The Error Ratio

In this subsection, we shall show that the minimum 2-star is a 1.577-approximation solution of the △PROCT problem. Assume $1/3 < q < 1/2$. We are going to construct a 2-star from the optimal solution, and then show the error ratio is a function of q. By choosing a suitable q, we can get the 1.577 error ratio.

Let $G = (V, E, w)$ and r be the input metric graph and the vertex weight. Also let $R = r(V)$ and T be the optimal tree. Let m be the vertex such that $\sum_{i \in V} r(i)d(T, i, m)$ is minimum among all vertices. If we root the tree at m then $r(T_x) \leq R/2 \ \forall x \neq m$. This property can be easily proved by contradiction and we do not show it here. We call such a vertex as the r-median of T. Root T at its r-median m and let $1/3 < q < 0.5$ be a specified real number. Consider all possible vertex x such that $r(T_x) \geq qR$ and $r(T_u) < qR \ \forall u \in V(T_x) - \{x\}$. Since $q > 1/3$, there are only three cases:

- there are two such vertices a and b;
- there is only one such vertex $a \neq m$;
- m is the only one such vertex.

For each case, we select two vertices. For the first case, a and b are selected. For the second case, a and m are selected, and the third case can be thought as a special case in which the two vertices are both m. Without loss of generality, assume the two vertices be a and b, and $M = SP(T, a, b) = (a = m_1, m_2, \dots, m_k = b)$ be the path on T. Let $d(T, u, M)$ denote the shortest distance from a vertex u to the path M on T. Also let U_i be the set of vertices connected to M at m_i. We have the following lemma:

Lemma 10. *$C_p(T) \geq 2\,(1-q)\,R \sum_x r(x)d(T, x, M) + 2q(1-q)R^2 w(M)$.*

Proof. For any vertex x, let $SB(x) = \{u | SP(T, x, u) \cap M = \emptyset\}$. Note that $r(SB(x)) \leq qR$. If $x \in U_i$ and $y \in U_j$, define $g(x, y) = d(T, m_i, m_j)$. Then,

$$\begin{aligned} C_p(T) &\geq \sum_x \sum_{y \notin SB(x)} r(x)r(y)\{d(T, x, M) + d(T, y, M) + g(x, y)\} \\ &\geq 2\,(1-q)\,R \sum_x r(x)d(T, x, M) + 2 \sum_{i<j} r(U_i)r(U_j)d(T, m_i, m_j) \end{aligned}$$

Without loss of generality, we assume $r(U_1) \geq r(U_k)$. For the second term,

$$\begin{aligned} 2 \sum_{i<j} r(U_i)r(U_j)d(T, m_i, m_j) &\geq 2r(U_1)r(U_k)w(M) + 2 \sum_{i=2}^{k-1} r(U_i)r(U_k)w(M) \\ &= 2w(M)r(U_k)\,(R - r(U_k)) \end{aligned}$$

Since $r(U_k) \geq qR$ and $q < 0.5$, $r(U_k)\,(R - r(U_k)) \geq q(1-q)R^2$ and complete the proof. □

Let $T^* = 2star(a, b, V - U_k, U_k)$ and $T^{**} = 2star(a, b, U_1, V - U_1)$. We show the following lemma:

Lemma 11. *$C_p(T^*) + C_p(T^{**}) \leq 4R \sum_{i \in V} r(i)d(T, i, M) + 2(1 - 2q^2)R^2 w(M)$.*

Proof. From Lemma 7,

$$\begin{aligned} C_p(T^*) \le\ & 2\sum_{i\notin U_k} r(i)(R-r(i))w(i,a) + 2\sum_{i\in U_k} r(i)(R-r(i))w(i,b) \\ & +2r(U_k)(R-r(U_k))w(a,b) \\ \le\ & 2R\sum_{i\notin U_k} r(i)w(i,a) + 2R\sum_{i\in U_k} r(i)w(i,b) + 2r(U_k)(R-r(U_k))w(a,b) \end{aligned}$$

Similarly,

$$C_p(T^{**}) \le 2R\sum_{i\in U_1} r(i)w(i,a) + 2R\sum_{i\notin U_1} r(i)w(i,b) + 2r(U_1)(R-r(U_1))w(a,b)$$

For $i \in U_1$, $w(i,a) \le d(T,i,M)$. Similarly, $w(i,b) \le d(T,i,M)\ \forall i \in U_k$. For $i \notin U_1 \cup U_k$, $w(i,a)+w(i,b) \le 2d(T,i,M)+w(M)$. So, we have

$$\begin{aligned} C_p(T^*)+C_p(T^{**}) \le\ & 4R\sum_i r(i)d(T,i,M) + 2R\sum_{i\notin U_1\cup U_k} r(i)w(M) \\ & +2r(U_k)(R-r(U_k))w(a,b) + 2r(U_1)(R-r(U_1))w(a,b) \\ \le\ & 4R\sum_i r(i)d(T,i,M) + 2(1-2q^2)R^2w(M) \end{aligned}$$

□

Lemma 12. There exists a 2-star, which is a 1.577-approximation solution for the △PROCT problem.

Proof. Trivially, T^* and T^{**} are both 2-stars. From Lemma 11 and 10, we have

$$\begin{aligned} \min\{C_p(T^*), C_p(T^{**})\} \le\ & 2R\sum_{i\in V} r(i)d(T,i,M) + (1-2q^2)R^2w(M) \\ \le\ & \max\{\frac{1}{1-q}, \frac{1-2q^2}{2q(1-q)}\}C_p(T) \end{aligned}$$

Therefore, the error ratio is $\max\{\frac{1}{1-q}, \left(1-2q^2\right)/\left(2q(1-q)\right)\}$ in which $1/3 < q < 1/2$. By setting $q = \frac{\sqrt{3}-1}{2} = 0.366$, we get the error ratio 1.577. □

4 The Sum-Requirement OCT Problem

In this section, we present a 2-approximation algorithm for the SROCT problem with general input. Our approximation uses the *shortest path tree*. Let m be a vertex. A shortest path tree rooted at m is a spanning tree T such that $d(T,i,m)$ equals the shortest path length between i and m on the original graph for any vertex i. The shortest path tree has been well studied and several efficient algorithms can be found in the literals, e.g. [1]. Our work is to show that there always exists a vertex m such that the shortest path tree rooted at m is a 2-approximation solution.

Definition 8. Let T be a tree and $e \in E(T)$. Assume T_1 and T_2 be the two trees resulted by deleting e from T. Define the routing load on edge e to be $l(T, e) = |V(T_1)|r(T_2) + |V(T_2)|r(T_1)$.

Let $G = (V, E, w)$ and r be the input graph and vertex weight. Assume $|V| = n$, $R = r(V)$. The following lemma comes directly from the definition and we omit the proof here.

Lemma 13. Let T be a tree. $C_s(T) \leq 2\sum_{i\in V(T)} (nr(i) + R)\, d(T, i, v)$, for any $v \in V(T)$.

In the following, T denotes the optimal spanning tree of the SROCT problem, and m_1 and m_2 are the median and r-median of T respectively. Also let $P = SP(T, m_1, m_2)$. The proof of the next lemma is ommited.

Lemma 14. $l(T, e) \geq nR/2$ for any edge $e \in SP(T, m_1, m_2)$.

We now give a lower bound of the optimal cost.

Lemma 15. $C_s(T) \geq \sum_{v\in V(T)} (nr(v) + R)\, d(T, v, P) + nRw(P)$.

Proof. For $u, v \in V$, let $B(u) = \{v|SP(T, u, v) \cap P \neq \emptyset\}$ and $w_P^T(u, v) = w(SP(T, u, v) \cap P)$. Note that $|B(u)| \geq n/2$ and $r(B(u)) \geq R/2$.

$$\begin{aligned}
C_s(T) &\geq \sum_u \sum_{v\in B(u)} (r(u) + r(v)) \left(d(T, u, P) + d(T, v, P) + w_P^T(u, v)\right) \\
&= 2\sum_u \left(\sum_{v\in B(u)} 1\right) r(u)d(T, u, P) + 2\sum_u \sum_{v\in B(u)} r(u)d(T, v, P) \\
&\quad +2\sum_u \sum_{v\in B(u)} r(v)w_P^T(u, v) \\
&\geq n\sum_u r(u)d(T, u, P) + R\sum_v d(T, v, P) + 2\sum_{e\in P} l(T, e)w(e) \\
&= \sum_{v\in V(T)} (nr(v) + R)\, d(T, v, P) + nRw(P)
\end{aligned}$$

□

Theorem 16. There exists a 2-approxiamtion algorithm with time complexity $O(n^3)$ for the SROCT problem.

Proof. Let Y^* and Y^{**} be the shortest path trees rooted at m_1 and m_2 respectively. Also let V_1 and V_2 denote the set of vertices connected to P at m_1 and m_2

respectively and $V_0 = V - V_1 - V_2$. By Lemma 13,

$$\begin{aligned}
C_s(Y^*)/2 &\le \sum_{i\in V} (nr(i)+R)\, d(Y^*, i, m_1) \\
&\le \sum_{i\in V_1} (nr(i)+R)\, d(T,i,m_1) + \sum_{i\in V_2} (nr(i)+R)\,(d(T,i,m_2)+w(P)) \\
&\quad + \sum_{i\in V_0} (nr(i)+R)\,\big(d(T,i,P) + w_P^T(i,m_1)\big) \\
&\le \sum_{i\in V} (nr(i)+R)\, d(T,i,P) + \sum_{i\in V_2} (nr(i)+R)\, w(P) \\
&\quad + \sum_{i\in V_0} (nr(i)+R)\, w_P^T(i,m_1)
\end{aligned}$$

Similarly

$$\begin{aligned}
C_s(Y^{**})/2 &\le \sum_{i\in V} (nr(i)+R)\, d(T,i,P) + \sum_{i\in V_1} (nr(i)+R)\, w(P) \\
&\quad + \sum_{i\in V_0} (nr(i)+R)\, w_P^T(i,m_2)
\end{aligned}$$

Since $w_P^T(i,m_1) + w_P^T(i,m_2) = w(P)$, we have

$$\begin{aligned}
\min\{C_s(Y^*), C_s(Y^{**})\} &\le \left(C_s(Y^*) + C_s(Y^{**})\right)/2 \\
\le 2\sum_{i\in V} (nr(i)+R)\, d(T,i,P) &+ 2nRw(P) \le 2C_s(T)
\end{aligned}$$

We have proved that there exists a shortest path tree which is a 2-approximation solution. By computing shortest path tree rooted at each vertex, we can find the shortest path tree with minimum cost. The result follows that all shortest path trees can be computed in $O(n^3)$ time. □

References

1. T.H. Coremen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithm*, the MIT Press, 1994. 412, 414
2. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Fransisco, 1979. 408
3. T. C. Hu, Optimum communication spanning tree, *SIAM J. Computing*, 3(3):188-195, 1974. 407
4. D.S. Johnson, J.K. Lenstra, and A.H.G. Rinnooy Kan, The Complexity of the network Design problem, *Networks*, 8:279-285, 1978. 408
5. R. Wong, Worst-case analysis of network design problem heuristics. *SIAM J. Algebraic Discrete Mathematics*, 1:51-63, 1980. 408
6. B. Y. Wu, G. Lancia, V. Bafna, K. M. Chao, R. Ravi, and C. Y. Tang, A polynomial time approximation scheme for minimum routing cost spanning trees, *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98)*, pp. 21-32, 1998. 408, 410

The Edge-Disjoint Paths Problem is NP-Complete for Partial *k*-Trees

Xiao Zhou and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University
Aoba-yama 05, Sendai 980-8579, Japan
{zhou,nishi}@ecei.tohoku.ac.jp

Abstract. Many combinatorial problems are NP-complete for general graphs, but are not NP-complete for partial k-trees (graphs of treewidth bounded by a constant k) and can be efficiently solved in polynomial time or mostly in linear time for partial k-trees. On the other hand, very few problems are known to be NP-complete for partial k-trees with bounded k. These include the subgraph isomorphism problem and the bandwidth problem. However, all these problems are NP-complete even for ordinary trees or forests. In this paper we show that the edge-disjoint paths problem is NP-complete for partial k-trees with some bounded k, say $k = 3$, although the problem is trivially solvable for trees.

1 Introduction

Many combinatorial problems are NP-complete for general graphs, and are unlikely to be solvable in polynomial time. However, many "natural" problems defined on unweighted graphs can be efficiently solved for partial k-trees (graphs of treewidth bounded by a constant k) in polynomial time or in linear time [1,2,4,6,23,26]. On the other hand, very few problems are known to be NP-complete for partial k-trees. These include the subgraph isomorphism problem and the bandwidth problem [7,10,15,20]. However, all these problems are NP-complete even for ordinary trees or forests, i.e. partial 1-trees [9].

The edge-disjoint paths problem asks whether there exist p pairwise edge-disjoint paths P_i, $1 \leq i \leq p$, connecting terminals s_i and t_i in a given graph G with p terminal pairs (s_i, t_i), $1 \leq i \leq p$, assigned to vertices of G. The vertex-disjoint paths problem is similarly defined. These problems come up naturally when analyzing connectivity questions or generalizing (integral) network flow problems. Another reason for the growing interest is the variety of applications, e.g. in VLSI-design and communication [12,13,18,21,22,25]. If $p = O(1)$, then the vertex-disjoint paths problem can be solved in polynomial time for any graph by Robertson and Seymour's algorithm based on their series of papers on graph minor theory [17]. The edge-disjoint paths problem on a graph G can be reduced in polynomial time to the vertex-disjoint paths problem on a new graph similar to the line graph of G. Therefore, the edge-disjoint paths problem can also be solved in polynomial time for any graph by the algorithm if $p = O(1)$. However, if p

is not bounded, then both the edge-disjoint and vertex-disjoint paths problems are NP-complete even for planar graphs [14,24].

A natural question is whether the vertex-disjoint and edge-disjoint paths problems can be efficiently solved for a restricted class of graphs, other than planar graphs, say partial k-trees. Indeed Scheffler showed that the vertex-disjoint paths problem can be solved in linear time for partial k-trees even if p is not bounded [19]. Frank obtained a necessary and sufficient condition for the existence of edge-disjoint paths in a class of planar graphs [8]. The result together with the algorithms in [13,25] yields a polynomial-time algorithm for a class of partial 2-trees, i.e., outer planar graphs. Zhou *et al.* showed that the edge-disjoint paths problem can be solved in polynomial time for partial k-trees if either $p = O(\log n)$ or the location of terminals satisfies some condition, where n denotes the number of vertices in a given partial k-tree [27]. However, it has not been known whether the edge-disjoint paths problem is NP-complete for partial k-trees if there is no restriction on the number of terminal pairs or the location of terminals.

In this paper we show that the edge-disjoint paths problem is NP-complete for partial 3-trees. This is the first example of problems which are efficiently solvable for trees and outer planar graphs but NP-complete for partial k-trees with some bounded $k \geq 2$.

The edge-disjoint paths problem is a decision problem, i.e., a YES-NO problem. The optimization version is the maximum edge-disjoint paths problem which asks to find a maximum number of edge-disjoint paths connecting terminal pairs. Garg *et al.* [11] showed that the maximum multicommodity integral flow problem is NP-hard for a tree with edge-capacities 1 or 2. Replace each edge of capacity 2 in the tree with a pair of parallel paths of length 2, then the resulting graph is an outer planar graph and hence is a partial 2-tree. Thus the result in [11] immediately implies that the maximum edge-disjoint paths problem is NP-hard for outer planar graphs or partial 2-trees. However, it does not imply our result because the NP-hardness of an optimization problem does not always imply the NP-completeness of the decision problem. Thus the complexities of edge-disjoint paths problems are similar to those of SAT problems: 2-SAT is polynomial-time solvable, MAX 2-SAT is NP-hard, and 3-SAT is NP-complete.

2 Terminology and Definitions

In this section we give some definitions. Let $G = (V, E)$ denote a graph with vertex set V and edge set E. The paper deals with *simple undirected* graphs without multiple edges or self-loops. Throughout the paper a path is assumed to be *simple*, that is, it does not pass through any vertex more than once. An edge joining vertices u and v is denoted by (u, v).

The class of *k-trees* is defined recursively as follows:

(a) A complete graph with k vertices is a k-tree.
(b) If $G = (V, E)$ is a k-tree and k vertices $v_1, v_2, \cdots, v_k$ induce a complete subgraph of G, then $G' = (V \cup \{w\}, E \cup \{(v_i, w) | 1 \leq i \leq k\})$ is a k-tree where w is a new vertex not contained in G.
(c) All k-trees can be formed with rules (a) and (b).

A graph is a *partial k-tree* if it is a subgraph of a k-tree. Thus a partial k-tree $G = (V, E)$ is a simple graph, and $|E| < kn$. In this paper we assume that k is a fixed constant.

A *tree-decomposition* of a graph $G = (V, E)$ is a tree $T = (V_T, E_T)$ with V_T a family of subsets of V satisfying the following properties [16]:

- $\bigcup_{X_i \in V_T} X_i = V$;
- for every edge $e = (v, w) \in E$, there is a node $X_i \in V_T$ with $v, w \in X_i$; and
- if node X_j lies on the path in T from node X_i to node X_l, then $X_i \cap X_l \subseteq X_j$.

The *width* of a tree-decomposition $T = (V_T, E_T)$ is $\max_{X_i \in V_T} |X_i| - 1$. The *treewidth* of graph G is the minimum width of a tree-decomposition of G, taken over all possible tree-decompositions of G. It is known that every graph with treewidth $\leq k$ is a partial k-tree, and conversely, that every partial k-tree has a tree-decomposition with width $\leq k$. Bodlaender has given a linear-time sequential algorithm to find a tree-decomposition of G with width $\leq k$ for fixed k [3].

3 The Edge-Disjoint Paths Problem

Our main result is the following theorem.

Theorem 1. *The edge-disjoint paths problem is NP-complete for partial* 3*-trees.*

In the remainder of this section we will give a proof of Theorem 1. Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of n Boolean variables. A *literal of $x_i \in X$* is either a Boolean variable x_i or its negation $\overline{x}_i$. We denote by *3-CNF* the set of Boolean formulas in a conjunctive normal form over the n variables in X with at most three literals per clause. For a Boolean formula $f \in$ 3-CNF, the *3-SAT problem* asks whether there is an assignment $\boldsymbol{a}$ of true-false values to the n variables such that $f(\boldsymbol{a})$ is true [5]. If there is an assignment $\boldsymbol{a}$ such that $f(\boldsymbol{a})$ is true, then we say that f is *satisfiable*. Clearly the edge-disjoint path problem is in NP [9]. Therefore it suffices to show that the 3-SAT problem can be reduced in polynomial time to the edge-disjoint paths problem for partial 3-trees.

For example, consider the following Boolean formula f with $n = 4$ variables and $m = 3$ clauses:

$$f = (x_1 + \overline{x}_2 + x_4)(x_2 + \overline{x}_3)(\overline{x}_1 + x_3 + \overline{x}_4).$$

As illustrated in Figure 1, we will construct a graph G_f which contains edge-disjoint paths connecting terminal pairs if and only if f is satisfiable. The formula f above is satisfiable for a true-false assignment $\boldsymbol{a}$ such

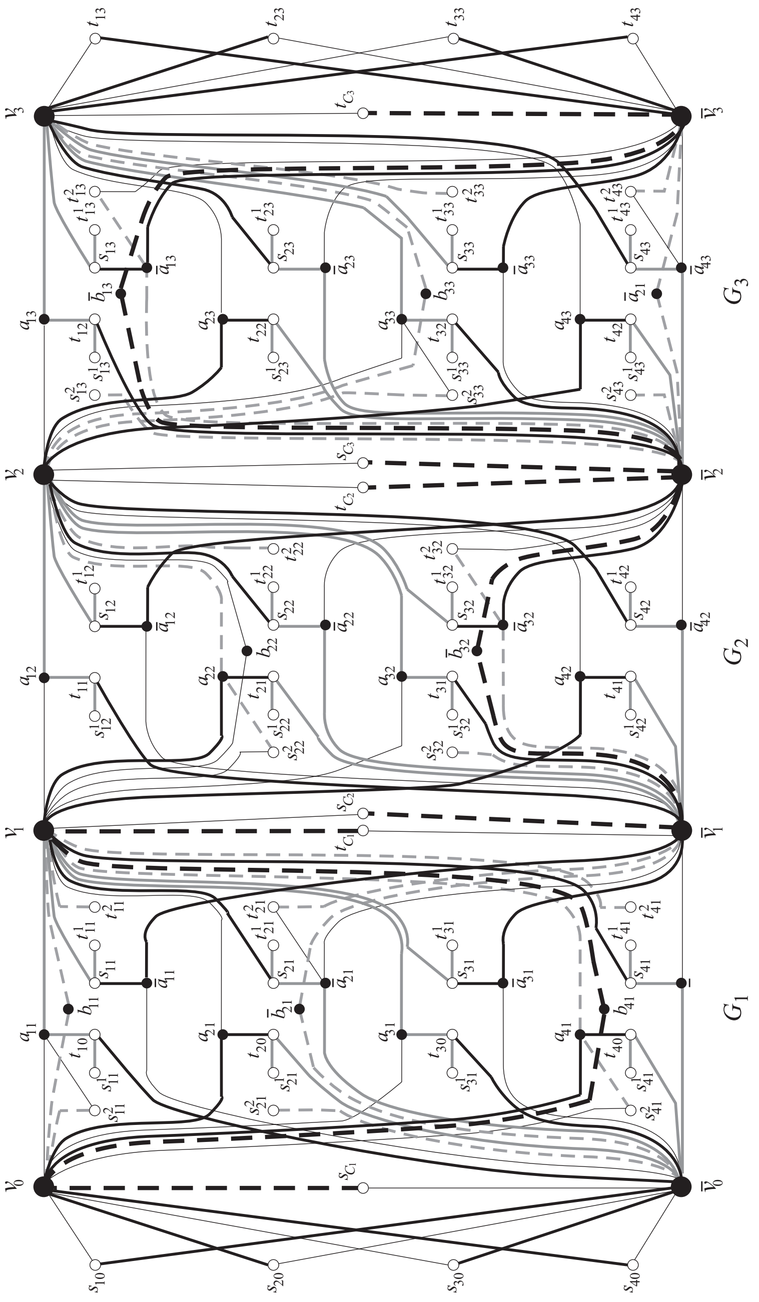


Fig. 1. Graph G_f and edge-disjoint paths.

that $x_1 = 0$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 1$, and G_f in Figure 1 has edge-disjoint paths connecting terminal pairs. In Figure 1 all terminals are drawn by white circles, and the edge-disjoint paths are drawn in thick solid or dotted lines colored black or gray.

Roughly speaking, G_f has a "grid structure" with n rows and $m+2$ columns. The ith row corresponds to variable x_i for each i, $1 \le i \le n$. The leftmost column, i.e. 0th column, denoted by G_0, contains n isolated vertices s_{i0}, $1 \le i \le n$. The rightmost column, i.e. $(m+1)$st column, denoted by G_{m+1}, contains n isolated vertices t_{im}, $1 \le i \le n$. The jth column, $1 \le j \le m$, denoted by G_j, corresponds to the jth clause C_j in f and contains n gadgets $G_{1j}, G_{2j}, \cdots, G_{nj}$, which will be defined later. For each j, $0 \le j \le m$, two consecutive columns G_j and G_{j+1} are connected through exactly two vertices v_j and $\overline{v}_j$, called *upper* or *lower connection vertices*, as illustrated in Figure 1. Thus the graph G_f constructed in this way has essentially a "ladder" structure instead of a "grid" structure, and is a partial 3-tree as we will observe later.

We now describe how to construct G_f in detail. Consider a Boolean formula $f \in$3-CNF with m clauses $C_1, C_2, \cdots, C_m$ and n variables in X. We write $x_i \in C_j$ and $\overline{x}_i \in C_j$ if clause C_j contains literals x_i and $\overline{x}_i$, respectively. One may assume that, for any variable x_i, $1 \le i \le n$, and any clause C_j, $1 \le j \le m$, exactly one of the following three cases occurs:

(a) $x_i \in C_j$ and $\overline{x}_i \notin C_j$,
(b) $x_i \notin C_j$ and $\overline{x}_i \in C_j$, and
(c) $x_i, \overline{x}_i \notin C_j$.

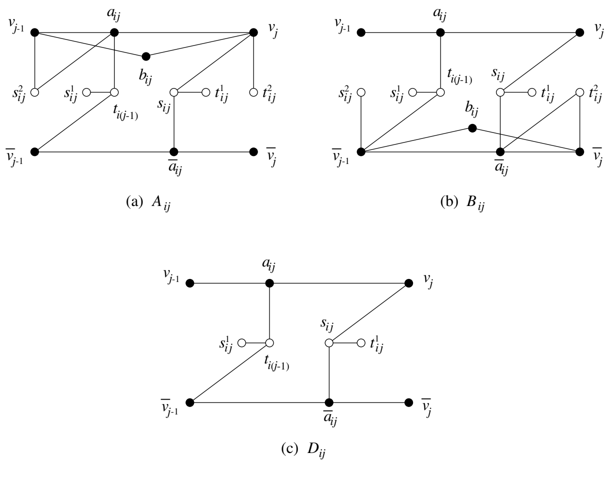


Fig. 2. Three types of gadgets A_{ij}, B_{ij} and D_{ij}.

For $1 \le i \le n$ and $1 \le j \le m$, we place a graph G_{ij}, called the *ij-gadget*, at the ith row and jth column in G_f. The construction of G_{ij} depends on which case among (a), (b) and (c) above occurs. Denote the graphs in Figure 2 (a), (b) and (c) by A_{ij}, B_{ij} and D_{ij}, respectively. Then we define G_{ij} as follows:

$$G_{ij} = \begin{cases} A_{ij} \text{ if } x_i \in C_j, \\ B_{ij} \text{ if } \overline{x}_i \in C_j, \text{ and} \\ D_{ij} \text{ if } x_i, \overline{x}_i \notin C_j. \end{cases}$$

The graph A_{ij} in Figure 2(a) has thirteen vertices: four outer vertices and nine inner ones. Two of the four outer vertices are v_{j-1} and $\overline{v}_{j-1}$ on the left side, and the other two are v_j and $\overline{v}_j$ on the right. Six of the nine inner vertices are terminals $t_{i(j-1)}$, s_{ij}, s^1_{ij}, t^1_{ij}, s^2_{ij} and t^2_{ij}. Both (s^1_{ij}, t^1_{ij}) and (s^2_{ij}, t^2_{ij}) are terminal pairs in G_{ij}. The terminal $t_{i(j-1)}$ in G_{ij} is paired with the terminal $s_{i(j-1)}$ in $G_{i(j-1)}$, and the terminal s_{ij} in G_{ij} is paired with the terminal t_{ij} in $G_{i(j+1)}$. Only the four outer vertices in A_{ij} are connected to vertices in other gadgets or connection vertices in G_f; none of the nine inner vertices in A_{ij} is connected to any vertex in other gadgets or any connection vertex in G_f. Similarly we construct B_{ij} and D_{ij} as illustrated in Figures 2(b) and (c). B_{ij} is isomorphic to A_{ij} rotated 180°. D_{ij} is obtained from A_{ij} by deleting vertices b_{ij}, s^2_{ij} and t^2_{ij}.

For each clause C_j, $1 \le j \le m$, we now construct the jth column G_j of graph G_f. The graph G_j has n rows ordered from the top to the bottom by $1, 2, \cdots, n$. For each i, $1 \le i \le n$, the ith row in G_j corresponds to the variable x_i, and contains G_{ij}. Identify all the n outer vertices v_{j-1} on the left sides of gadgets $G_{1j}, G_{2j}, \cdots, G_{nj}$ as an upper connection vertex v_{j-1}; identify all the n outer vertices $\overline{v}_{j-1}$ on the left sides as a lower connection vertex $\overline{v}_{j-1}$; identify all the n outer vertices v_j on the right sides as an upper connection vertex v_j; and identify all the n outer vertices $\overline{v}_j$ on the right sides as a lower connection vertex $\overline{v}_j$. Add to G_j a terminal pair (s_{C_j}, t_{C_j}) as follows. (See Figure 1.) If C_j contains an affirmative literal, that is, $x_i \in C_j$ for some i, $1 \le i \le n$, then join s_{C_j} with v_{j-1} and join t_{C_j} with v_j. If C_j contains a negative literal, that is, $\overline{x}_i \in C_j$ for some i, $1 \le i \le n$, then join s_{C_j} with $\overline{v}_{j-1}$ and join t_{C_j} with $\overline{v}_j$. Thus the degrees of both vertices s_{C_j} and t_{C_j} are one or two.

The graph G_0, i.e., the 0th column of G_f, consists of n vertices s_{i0}, $1 \le i \le n$, together with two vertices v_0 and $\overline{v}_0$, where s_{i0}, $1 \le i \le n$, is connected to v_0 and $\overline{v}_0$ as illustrated in Figure 1. Similarly, we construct G_{m+1}.

We now construct the whole graph G_f by connecting the $m + 2$ graphs $G_0, G_1, \cdots, G_{m+1}$ in cascade as illustrated in Figure 1; any two consecutive columns G_j and G_{j+1}, $0 \le j \le m$, are connected through exactly two connection vertices v_j and $\overline{v}_j$. This completes the construction of G_f, which essentially has a "ladder" structure.

Let N be the number of vertices in G_f, then clearly $N = \Theta(mn)$. On the other hand, G_f has the following p terminal pairs:

(a) $(m+1)n$ pairs (s_{ij}, t_{ij}) for all i and j such that $1 \le i \le n$ and $0 \le j \le m$;
(b) m pairs (s_{C_j}, t_{C_j}) for all j such that $1 \le j \le m$;
(c) mn pairs (s^1_{ij}, t^1_{ij}) for all j and j such that $1 \le i \le n$ and $1 \le j \le m$; and

(d) at most $3m$ pairs (s_{ij}^2, t_{ij}^2) for all i and j such that $1 \le i \le n$, $1 \le j \le m$ and x_i or $\overline{x}_i \in C_j$.

Hence $p \le (m+1)n + m + mn + 3m = O(N)$. Thus, from f, one can construct the graph G_f and assign the p terminal pairs to vertices in G_f in polynomial time. We denote by P_{ij} the paths for pairs in (a), by P_{C_j} those for (b), by P_{ij}^1 those for (c), and by P_{ij}^2 those for (d).

The following lemma holds.

Lemma 1. *G_f is a partial 3-tree.*

Thus it suffices to prove the following lemma.

Lemma 2. *G_f has edge-disjoint paths connecting terminal pairs if and only if f is satisfiable.*

In the remainder of this section we will give a proof of Lemma 2. We first have the following lemma.

Lemma 3. *Assume that G_f has edge-disjoint (simple) paths connecting terminal pairs. Then none of the paths passes through v_j and $\overline{v}_{j'}$ for some j and j', $0 \le j, j' \le m$.*

A path P connecting a terminal pair is called an *inner path* if the two terminals and all the edges of P are contained in a single gadget. Thus the terminal pair of an inner path is (s_{ij}^1, t_{ij}^1) or (s_{ij}^2, t_{ij}^2) for some i and j, $1 \le i \le n$ and $1 \le j \le m$. Such an inner-path P is called a *G_{ij}-inner-path.* A path P is called a *G_{ij}-through-path* if both of the terminals of P are outside G_{ij} and P passes through at least one of the edges in G_{ij}. Lemma 3 implies that a G_{ij}-through-path P passes either through v_{j-1} and v_j or through $\overline{v}_{j-1}$ and $\overline{v}_j$. In the former case P is called an *upper G_{ij}-through-path*, and in the latter case P is called a *lower G_{ij}-through-path.* We then have the following three lemmas.

Lemma 4. *Assume that G_f has edge-disjoint paths connecting terminal pairs. If a path P among them is a G_{ij}-through-path for some i and j, $1 \le i \le n$ and $1 \le j \le m$, then P is an upper or lower G_{ij}-through-path.*

Lemma 5. *Assume that G_f has edge-disjoint paths connecting terminal pairs, and that a path P among them is a G_{ij}-through-path for some i and j, $1 \le i \le n$ and $1 \le j \le m$. Then the following* (a) *and* (b) *hold:*

(a) *$G_{ij} = A_{ij}$ if and only if P is an upper G_{ij}-through-path; and*
(b) *$G_{ij} = B_{ij}$ if and only if P is a lower G_{ij}-through-path.*

Lemma 6. *Assume that G_f has edge-disjoint paths connecting terminal pairs, and that P_{C_j} is a G_{ij}-through-path for some i and j, $1 \le i \le n$ and $1 \le j \le m$. Then the following* (a) *and* (b) *hold:*

(a) *if $G_{ij} = A_{ij}$, then there is a variable $x_{i'}$, $1 \le i' \le n$, such that $x_{i'} \in C_j$ and $P_{i'0}$ passes through v_0; and*

(b) *if $G_{ij} = B_{ij}$, then there is a variable $x_{i'}$, $1 \le i' \le n$, such that $\overline{x}_{i'} \in C_j$ and $P_{i'0}$ passes through $\overline{v}_0$.*

We are now ready to give a proof of Lemma 2.

Proof of Lemma 2 $\Rightarrow$: Suppose that G_f has edge-disjoint paths connecting terminal pairs. Let $\boldsymbol{a}$ be an assignment of true-false values to the n variables x_i, $1 \le i \le n$, such that x_i is true if P_{i0} passes through vertex v_0, and x_i is false if P_{i0} passes through vertex $\overline{v}_0$. Then we claim that $f(\boldsymbol{a})$ is true.

For each clause C_j, $1 \le j \le m$, P_{C_j} passes through G_{ij} for some i, $1 \le i \le n$, and hence P_{C_j} is a G_{ij}-through-path. Then, by Lemma 4 P_{C_j} is an upper or lower G_{ij}-through-path, and hence by Lemma 5 $G_{ij} = A_{ij}$ or B_{ij}.

Consider first the case $G_{ij} = A_{ij}$. Then, by Lemma 6(a), $x_{i'} \in C_j$ and $P_{i'0}$ passes through vertex v_0 for some i', $1 \le i' \le n$. Thus the value of variable $x_{i'}$ is true, and hence C_j is true.

Consider next the case $G_{ij} = B_{ij}$. By Lemma 6(b), $\overline{x}_{i'} \in C_j$ and $P_{i'0}$ passes through vertex $\overline{v}_0$ for some i', $1 \le i' \le n$. Thus the value of variable $x_{i'}$ is false, the value of literal $\overline{x}_{i'}$ in C_j is true, and hence C_j is true.

Thus all clauses C_j of f are true, and hence $f(\boldsymbol{a})$ is true.

$\Leftarrow$: Suppose that f is satisfiable. Then there is an assignment $\boldsymbol{a}$ such that $f(\boldsymbol{a})$ is true. We shall show that G_f has edge-disjoint paths connecting all terminal pairs.

We first find paths P_{ij} for all i and j, $1 \le i \le n$ and $0 \le j \le m$, as follows. If $\boldsymbol{a}$ assigns variable x_i, $1 \le i \le n$, true, then we make each path P_{ij}, $0 \le j \le m-1$, pass through s_{ij}, v_j, $a_{i(j+1)}$ and t_{ij}, and make P_{im} pass through s_{im}, v_m and t_{im}. If $\boldsymbol{a}$ assigns variable x_i, $1 \le i \le n$, false, then we make P_{i0} pass through s_{i0}, $\overline{v}_0$ and t_{i0}, and make each path P_{ij}, $1 \le j \le m$, pass through s_{ij}, $\overline{a}_{ij}$, $\overline{v}_j$ and t_{ij}.

We next find paths P^1_{ij} for all i and j, $1 \le i \le n$ and $1 \le j \le m$, as follows. If $\boldsymbol{a}$ assigns variable x_i, $1 \le i \le n$, true, then we make each path P^1_{ij}, $1 \le j \le m$, pass through s^1_{ij}, $t_{i(j-1)}$, $\overline{v}_{j-1}$, $\overline{a}_{ij}$, s_{ij} and t^1_{ij}. If $\boldsymbol{a}$ assigns variable x_i, $1 \le i \le n$, false, then we make each path P^1_{ij}, $1 \le j \le m$, pass through s^1_{ij}, $t_{i(j-1)}$, a_{ij}, v_j, s_{ij} and t^1_{ij}.

We next find paths P^2_{ij} for all i and j such that $1 \le i \le n$, $1 \le j \le m$ and x_i or $\overline{x}_i \in C_j$, as follows. There are the following two cases.

Case 1: $x_i \in C_j$.

If x_i is true, then we make P^2_{ij} pass through s^2_{ij}, a_{ij}, v_j and t^2_{ij}; otherwise, we make P^2_{ij} pass through $s^2_{ij}, v_{j-1}, b_{ij}, v_j$ and t^2_{ij}.

Case 2: $\overline{x}_i \in C_j$.

If x_i is true, then we make P^2_{ij} pass through $s^2_{ij}, \overline{v}_{j-1}, \overline{b}_{ij}, \overline{v}_j$ and t^2_{ij}; otherwise, we make P^2_{ij} pass through $s^2_{ij}, \overline{v}_{j-1}, \overline{a}_{ij}$ and t^2_{ij}.

We finally find paths P_{C_j}, $1 \le j \le m$, as follows. At least one of the three literals in C_j, say l_{ij}, is true under the assignment $\boldsymbol{a}$, where l_{ij} is either x_i or $\overline{x}_i$. Then either $l_{ij} = x_i$ and x_i is true or $l_{ij} = \overline{x}_i$ and x_i is false. For the former case, we make P_{C_j} pass through $s_{C_j}, v_{j-1}, b_{ij}, v_j$ and t_{C_j}. For the latter case, we make P_{C_j} pass through $s_{C_j}, \overline{v}_{j-1}, \overline{b}_{ij}, \overline{v}_j$ and t_{C_j}.

Thus we have proved that G_f has edge-disjoint paths connecting terminal pairs. □

4 Conclusion

In this paper we showed that the edge-disjoint paths problem is NP-complete for partial 3-trees. Since the graph G_f constructed by our reduction has a bounded pathwidth, our reduction implies that the edge-disjoint paths problem is NP-complete for the class of graphs with bounded pathwidth. Therefore the maximum edge-disjoint paths problem is NP-hard for the same class. It should be noted that the graph constructed by the reduction in [11] has an unbounded pathwidth.

Zhou *et al.* proved the following fact: the edge-disjoint paths problem can be solved in polynomial time for a partial k-tree G if the augmented graph G^+ obtained from G by adding p edges (s_i, t_i), $1 \leq i \leq p$, remains to be a partial k-tree [27]. The result in this paper does not conflict with the fact above, because the augmented graph G_f^+ of G_f is not always a partial k-tree with bounded k.

A class of tractable problems for partial k-trees has been characterized in terms of the monadic second order logic [1,2,4,6]. It remains open to characterize a class of intractable problems, including the edge-disjoint paths problem, the subgraph isomorphism problem, and the bandwidth problem.

Acknowledgment

Thanks to A. Frank and an anonymous referee for pointing out some references.

References

1. S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the Association for Computing Machinery*, 40(5):1134–1164, 1993. 417, 425
2. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. 417, 425
3. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996. 419
4. R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992. 417, 425
5. S.A. Cook. The complexity of theorem-proving procedures. In *Proc. of 3th Symp. Theory of Comp. Association for Computing Machinery*, pages 151–158, 1971. 419
6. B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990. 417, 425
7. A. Dessmark, A. Lingas, and A. Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. In *Proc. of the Fourth Europian Symposium on Algorithms, Lect. Notes in Computer Science, Springer-Verlag*, volume 1136, pages 501–513, 1996. 417

8. A. Frank. Edge-disjoint paths in planar graphs. *Journal of Combinatorial Theory, Series B*, 39:164–178, 1985. 418
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York, 1979. 417, 419
10. A. Gupta and N. Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoretical Computer Science*, 164:287–297, 1996. 417
11. N. Garg, V.V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997. 418, 425
12. J. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proc. ACM Symp. Theory of Computing*, pages 26–35, 1995. 417
13. K. Matsumoto, T. Nishizeki, and N. Saito. An efficient algorithm for finding multicommodity flows in planar networks. *SIAM J. Comput.*, 14(2):289–302, 1985. 417, 418
14. M. Middendorf and F. Pfeiffer. On the complexity of disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993. 418
15. J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108:343–364, 1992. 417
16. N. Robertson and P.D. Seymour. Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986. 419
17. N. Robertson and P.D. Seymour. Graph minors XIII. The disjoint paths problem. *J. of Combin. Theory, Series B*, 63(1):65–110, 1995. 417
18. H. Ripphausen-Lipa, D. Wagner, and K. Weihe. Efficient algorithms for disjoint paths in planar graphs. *DIMACS Series in Discrete Math. and Theoretical Computer Science*, 20:295–354, 1995. 417
19. P. Scheffler. A practial linear time algorithm for disjoint paths in graphs with bound tree-width. Technical Report 396, Dept. Mathematics, Technische Universität Berlin, 1994. 418
20. M. Sysło. NP-complete problems on some tree-structured graphs: a review. In *Proc. WG'83 International Workshop on Graph Theoretic Concepts in Computer Science*, pages 478–489, Univ. Verlag Rudolf Trauner, Linz, West Germany, 1983. 417
21. H. Suzuki, A. Ishiguro, and T. Nishizeki. Edge-disjoint paths in a grid bounded by two nested rectangles. *Discrete Applied Mathematics*, 27:157–178, 1990. 417
22. H. Suzuki, , T. Nishizeki, and N. Saito. Algorithms for multicommodity flows in planar graphs. *Algorithmica*, 4:471–501, 1989. 417
23. J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10:529–550, 1997. 417
24. J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Appl. Math.*, 61:83–90, 1995. 418
25. D. Wagner and K. Weihe. A linear-time algorithm for edge-disjoint paths in planar graphs. *Combinatorica.*, 15:135–150, 1995. 417, 418
26. X. Zhou, S. Nakano, and T. Nishizeki. Edge-coloring partial k-trees. *Journal of Algorithms*, 21:598–617, 1996. 417
27. X. Zhou, S. Tamura, and T. Nishizeki. Finding edge-disjoint paths in partial k-trees. In *Proc. of the Seventh International Symposium on Algorithms and Computation, Lect. Notes in Computer Science, Springer-Verlag*, volume 1178, pages 203–212, 1996. 418, 425

Inapproximability Results for Guarding Polygons without Holes[⋆]

Stephan Eidenbenz

Institute for Theoretical Computer Science, ETH 8092 Zürich, Switzerland
eidenben@inf.ethz.ch

Abstract. The three art gallery problems Vertex Guard, Edge Guard and Point Guard are known to be NP-hard [8]. Approximation algorithms for Vertex Guard and Edge Guard with a logarithmic ratio were proposed in [7]. We prove that for each of these problems, there exists a constant $\epsilon > 0$, such that no polynomial time algorithm can guarantee an approximation ratio of $1 + \epsilon$ unless $P = NP$. We obtain our results by proposing gap-preserving reductions, based on reductions from [8]. Our results are the first inapproximability results for these problems.

1 Introduction and Problem Definition

Guarding polygons is a variant of the art gallery problem, which asks how many guards are needed to see every point in the interior of a polygon P given as a linked list of n points in the $x-y$-plane. Polygon guarding problems are classified as to where the guards may be positioned, what kind of guards can be used, whether only the boundary or all of the interior of the polygon should be seen from at least one guard, and assumptions are also made on certain properties of the input polygon. In this paper, we assume that the input polygons are simple, i.e., such that no two nonconsecutive edges intersect. A point *sees* some other point, if the line segment connecting the two points does not intersect the exterior of the polygon P.

Many results are known concerning upper and lower bounds on the number of guards needed. Comparatively few papers study the computational complexity of art gallery problems. Surveys on the general topic of art galleries include [11,13], and [14]. [9] contains an overview of what is known about the computational complexity of several art gallery problems. As for the computational complexity of art gallery problems, it is known that the problem of covering a polygon with holes with a minimum number of convex polygons or star-shaped polygons is NP-hard [10]. The latter problem is equivalent to the Point Guard problem to be defined later. These problems remain NP-hard even if the input polygon has no holes (for convex polygons [4] and for star-shaped polygons [8]). The two problems Vertex Guard and Edge Guard (to be defined later) are NP-hard for polygons without holes [8]. Point Guard, Vertex Guard and Edge Guard

⋆ This work is partially supported by the Swiss National Science Foundation

for polygons with holes cannot be approximated by any polynomial algorithm within a factor $\frac{1-\epsilon}{28}\ln n$ for any $\epsilon > 0$ unless $NP \subseteq TIME(n^{O(\log\log n)})$ [6]. Finally, approximation algorithms for Vertex Guard and Edge Guard, which achieve approximation ratios of $O(\log n)$, exist [7]. The approximation algorithms work for polygons with and without holes.

We study the following problems:

- Vertex Guard(VG):
  Given a polygon P (without holes) with n vertices, find a minimum subset S of vertices such that every point on the boundary of the polygon P can be seen from at least one vertex in S. We say that guards are placed at the vertices in S. The vertices in S are called vertex guards.
- Edge Guard (EG):
  Given a polygon P (without holes) with n vertices, find a minimum subset S of edges, i.e. line segments of the polygon, such that every point on the boundary of the polygon P can be seen from at least one point on an edge in S. The edges in S are called edge-guards.
- Point Guard (PG):
  Given a polygon P (without holes) with n vertices, find a minimum subset S of points in the interior of the polygon such that every point on the boundary of the polygon P can be seen from at least one point in S. The points in S are called point-guards.

Note that our definitions differ from the corresponding definitions in [8] because in [8] the guards must see all of the interior of the polygon rather than only the boundary. It will be easy to see that our results carry over to these problems.

Since these problems are NP-hard, we would like to know how well they can be approximated by polynomial time algorithms. Urrutia points out that such results are needed [14]. When trying to determine the approximation properties of a problem, we always have two options; one of them is to find approximation algorithms that achieve a certain approximation ratio, as has been done for VG and EG [7]. The other option is to find lower bounds on the approximation ratio achievable, which is what we pursue in this paper. The result of this paper is that VG, EG and PG for polygons without holes are APX-hard, which means that for each of these problems, there is a constant $\epsilon > 0$ such that an approximation ratio of $1+\epsilon$ cannot be guaranteed by any polynomial time algorithm, unless $NP = P$. (See [3] for an introduction to the class APX.)

We prove the results by describing a reduction from 5-Occurrence-3-Sat, which is the version of 3-SAT with each clause containing at most three literals and with each variable appearing in at most five literals. 5-Occurrence-3-Sat is $MAXSNP$-complete [12], which means that there is a constant $\gamma > 0$ such that no polynomial time algorithm can guarantee an approximation-ratio of $1+\gamma$ for 5-Occurrence-3-Sat, unless $NP = P$ [2]. Our reduction follows the lines of the reductions in [8]. We show that our reduction is gap-preserving, using a technique introduced in [2].

In Sect. 2, we propose a construction that takes a 5-Occurrence-3-Sat instance as input and yields a polygon without holes, which is a PG-instance.

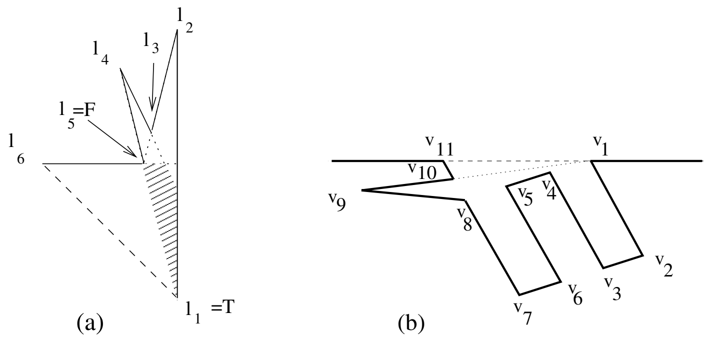


Fig. 1. (a) Literal Pattern; (b) Variable Pattern

The transformation of the solution is described in Sect. 3. We analyze the reduction and obtain our main result in Sect. 4. Section 5 contains some concluding remarks.

2 Construction of the Reduction

We present the construction for PG. Suppose we are given an instance I of 5-Occurrence-3-Sat. Let I consist of n variables $x_1, \cdots, x_n$ and of $m \leq \frac{5}{3}n$ clauses $c_1, \cdots, c_m$. We construct a polygon P that can be used in the reduction.

For every literal y_j, which is either x_j or $\neg x_j$, we construct a "literal pattern" as shown in Fig. 1(a). The literal pattern is the polygon defined by the points $l_1, \cdots, l_6$. The edge from l_6 to l_1 is not part of the final polygon, but serves as an interface to the outside of the literal pattern. All other edges in the literal pattern are part of the final polygon. The points l_4, l_5, l_1 lie on a straight line. Therefore, a guard at point l_1 or point l_5 sees all of the interior of the literal pattern. The final construction is such that a guard at point l_1 implies that the literal is true and such that a guard at point l_5 implies that the literal is false. We, therefore, call point l_1 simply T; similarly, l_5 is called F. For a finite number of guards, in order to completely see the whole literal pattern of Fig. 1(a), at least one guard must be inside the literal pattern. (If point l_4 of the pattern can be seen from a guard outside the pattern, which is possible by a guard on the line through l_4 and l_1, then in order to see the points on the segment from l_4 to l_3 from outside the pattern, an infinite number of guards is needed.)

For every clause c_i consisting of the literals y_j, y_k and y_l, we construct a "clause junction" as shown in Fig. 2. The clause junction is the polygon starting at point p_1 and moving along the solid line through p_2, p_3, p_4, the three literal patterns and ending at point p_8. The line from p_8 to p_1 is not part of the final polygon, but, again, serves as an interface of the clause junction to the outside. At least three guards are needed to completely see the clause junction in Fig. 2: one guard for each literal pattern. At least one of these three guards needs to be

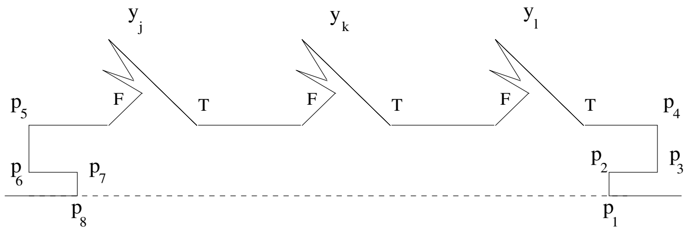


Fig. 2. Clause Junction

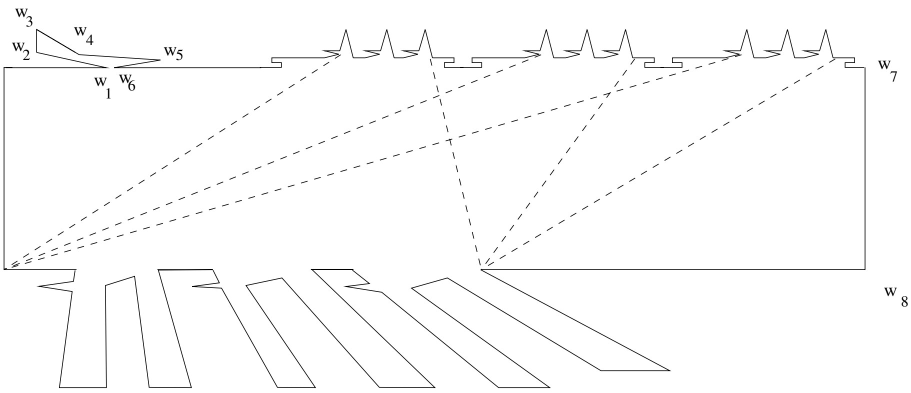


Fig. 3. Putting the pieces together

at point $l_1(= T)$ of the corresponding literal pattern in order to see point p_6 of the clause junction.

For every variable x_j, we construct a “variable pattern” as shown in Fig. 1(b). The variable pattern is the polygon defined by points $v_1, \cdots, v_{11}$. The edge from v_1 to v_{11} is not part of the final polygon, but serves as an interface. We call the polygon defined by the points v_1, v_2, v_3, v_4 the *right leg* of the variable pattern and the polygon defined by v_5, v_6, v_7, v_{11} the *left leg* of the variable pattern. The polygon formed by v_8, v_9, v_{10} is called the *triangle* of the variable pattern. The points v_9, v_{10}, v_1 lie on a straight line. Therefore, for a finite number of guards, in order to completely see the triangle (formed by the three points v_8, v_9, v_{10}), at least one guard must be inside the variable pattern. In the final polygon, this guard sits at point v_1, if the variable is assigned the value true, and it sits at point v_5, if the variable is false.

We put all pieces together as shown in Fig. 3. A guard at point w_1 sees all the legs of the variable patterns. The points w_3, w_4, w_1 are in a straight line. For a finite number of guards, in order to completely see the triangle w_1, w_2, w_3, at least one guard must lie inside the ear-like feature defined by the polygon $w_1, \cdots, w_6$. Finally, we construct for each literal y_j in each clause two “spikes” as shown in Fig. 4. Figure 4 (a) is for the case, when literal y_j is equal to x_j; Fig-

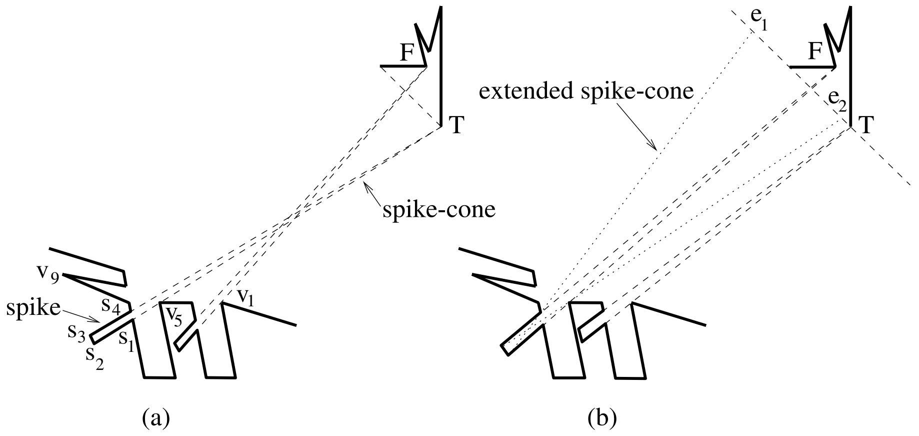


Fig. 4. (a): Spikes, when literal y_j is equal to x_j; (b): Spikes, when literal y_j is equal to $\neg x_j$

ure 4 (b) is for the case, when literal y_j is equal to $\neg x_j$. The points s_2, s_1 and $l_5 (= F)$ of the corresponding literal pattern are in a straight line, as are s_3, s_4 and l_5. A *spike* is a polygon defined by points s_1, s_2, s_3, s_4. The corresponding *spike-cone* is defined as the triangle s_1, s_4, l_1 or s_1, s_4, l_5. (Note that $l_1 = T$ and $l_5 = F$.) The points s_2, s_4, e_1 are in a straight line as are points s_3, v_1, e_2. We call the polygon s_3, v_1, e_2, e_1 the *extended spike-cone* of the corresponding spike (see Fig. 4 (b)). For a guard outside the variable pattern, in order to see point s_2 of a spike, it is a necessary condition that this guard lie in the corresponding extended spike-cone. We, therefore, call point s_2 of each spike the *distinguished point* of the spike. Note that the spikes may have to be very thin (thinner than indicated in Fig. 4), since up to five spikes must fit into each leg without intersecting each other.

In order to ensure that at no point between the variable and the literal patterns a guard sees the distinguished points of three or more spikes that belong to three different legs of variable patterns, we construct the polygon in such a way that no three extended spike-cones of different legs of variable patterns intersect. This restriction forces us to give a more detailed description of the whole construction.

First, fix the points $l_1 (= T)$ and $l_5 (= F)$ of each literal pattern on a horizontal line with constant distance between them. (We will move the F points above the line at the end of the construction.) For each l_1 and l_5 we fix the point e_1 at distance a to the left of the point l_1 or l_5. We also fix point e_2 at distance $2a$ to the right of the point l_1 or l_5 with a a constant as indicated in Fig. 5 (a). The constant a has to be small enough, in order to avoid that the extended spike-cones of two neighboring points l_1 and l_5 intersect at the horizontal line of the literal patterns, and in order to avoid that spikes in the same leg of a variable pattern intersect. Choose w_1 at a constant distance to the left of the

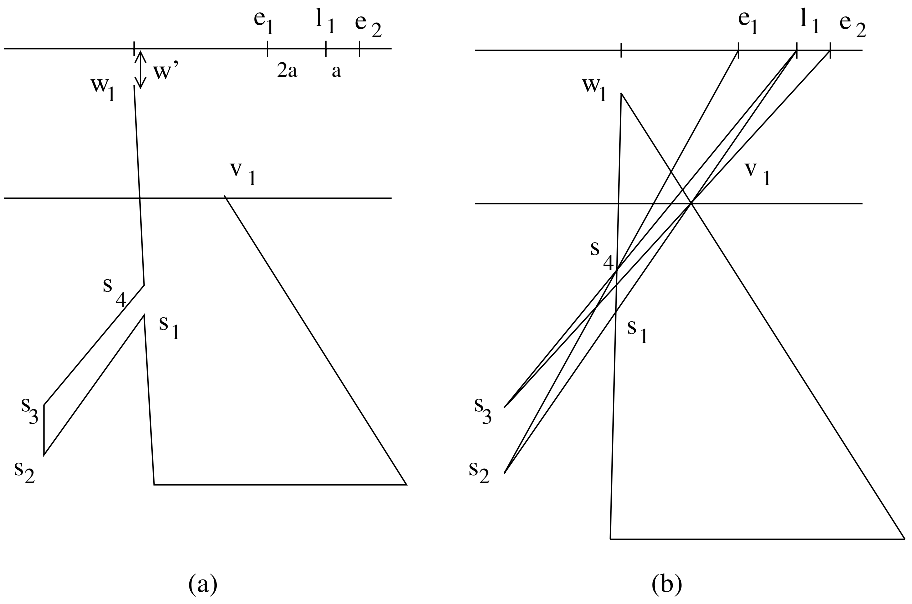


Fig. 5. Construction of the right leg

leftmost l_5 (of the leftmost literal) and at a constant distance w' below the line of the literal patterns.

Assume that the variable patterns for the variables $x_1, \cdots, x_{i-1}$ have already been constructed. We show how to construct the next variable pattern for variable x_i. We determine point v_1 of the variable pattern as follows: We determine the rightmost point l_5 or l_1 of the at most five literal patterns that are literals of variable x_i, from which a spike must be constructed in the right leg of the variable pattern. W. l. o. g. assume that this point is some point l_1. We draw a line from point e_2 (with respect to l_1) through the point S, which is the point with largest y-value of all points where two extended spike-cones intersect. Let v_1' be the point, where this line and the horizontal line of the variable patterns intersect. Now, let v_1 be at some constant distance to the left of v_1'. We now construct the right leg of the variable pattern as indicated in Fig. 5 (a), which shows the right leg with the top-most spike. Figure 5 (b) shows how (a) is constructed. We describe this step by step. Once point v_1 has been determined, draw a line segment from point e_2 through v_1 and stop at a certain, fixed x-distance from v_1. This yields point s_3. Draw a line segment from l_1 through v_1 and stop at the same x-distance from v_1, which is point s_2. Draw a line segment from s_3 to l_1. Then, draw a line segment from s_2 to e_1. Point s_4 is the intersection point of these two line segments. Finally, draw the leg of the variable pattern by drawing a line segment from w_1 through v_1 and a line segment from w_1 through s_4, which yields point s_1. We continue with the remaining spikes of the leg. The remaining spikes are drawn with x-distances from points s_4 to points s_2 always the same.

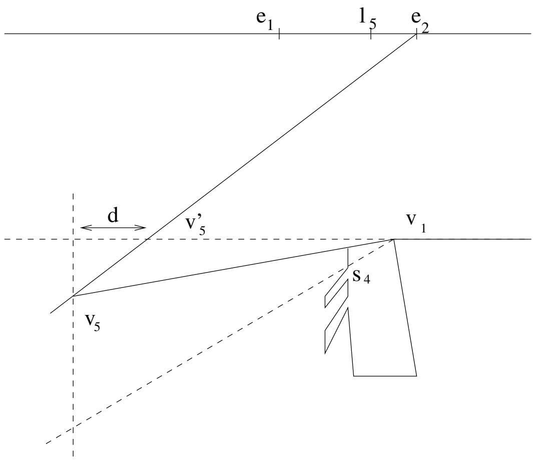


Fig. 6. The left leg

We construct the left leg of the variable pattern accordingly, except for point v_5. It is shown in Fig. 6 how to obtain v_5; v_5 is the intersection point of a vertical line at a constant distance d from v'_5 with either the extended spike-cone borderline from e_2 through v'_5 or with the line from v_1 through point s_4 (of the top-most spike) of the first leg. v'_5 is obtained using the same procedure as for point v_1.

Once we have constructed all variable patterns, we need to construct the literal patterns as shown in Fig. 7. In a similar method as used for the left leg of the variable pattern, we move point l_5 upwards. d' is a constant. This operation can be performed in such a manner that the extended spike-cone remains unchanged (or only "shrinks" in size) and that the spike-cone only changes in a way, which results in a smaller opening of the spike in the variable pattern. Finally, we construct the clause junctions as shown in Fig. 8. The clause junctions are constructed such that no spike-cone intersects with the polygon boundary. Distance b is defined to be the minimum of the y-distance of the point with maximum y-value among all points, where two extended spike-cones intersect, from the horizontal line of the literal patterns, and the distance w' (see Fig. 5). We complete the construction as indicated in Fig. 3; points w_1, w_6 and p_8 of the leftmost clause junction are in a straight line.

An analysis reveals that the coordinates of all points can be computed in polynomial time; some coordinates require a polynomial number of bits. (A similar analysis can be found in full detail in [5].) Therefore, the construction is polynomial in the size of the input.

3 Transformation of the Solution

We describe how to obtain an assignment of the variables of the satisfiability instance, given a feasible solution of the corresponding PG instance.

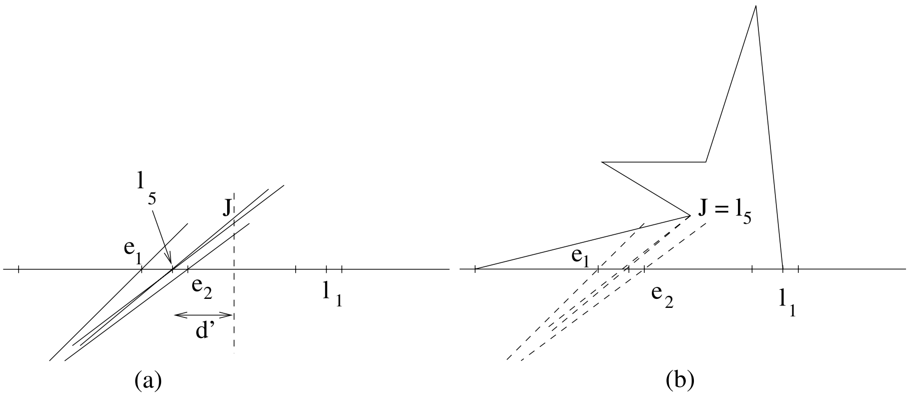


Fig. 7. Construction of the literal patterns

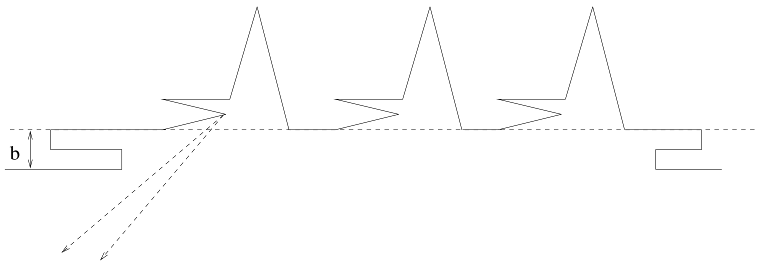


Fig. 8. Construction of the clause junctions

- Determine which guard is inside the polygon $w_1, \cdots, w_6$ (see Fig. 3) and move this guard to w_1.
- For each literal pattern, determine which guard sees point l_4 (and is inside the literal pattern) and move it to the closer one of the two points $l_1(=T)$ or $l_5(=F)$ of the literal pattern. If there is a guard at both points $l_1(=T)$ and $l_5(=F)$ of the literal pattern, move the guard at $l_5(=F)$ along the spike-cone to the point v_1 or v_5 of the corresponding variable pattern. If there is more than one guard at point $l_1(=T)$ or $l_5(=F)$, move all but one guard along the spike-cone to the variable pattern.
- For each clause junction, move any guard that sees point p_6 of the junction to the closest point T of any literal pattern in the clause junction.
- For each variable pattern, move the guard that sees point v_9 of the variable pattern to the closest of the two points v_1 and v_5 of the variable pattern. If there are additional guards in the area of the variable pattern, move them to v_1 or v_5 depending upon which leg of the variable pattern they are in.
- Move all guards that lie in an extended spike-cone, but not at point l_1 or l_5 of a literal pattern to point v_1 or v_5 of the corresponding variable pattern. If a guard lies in an intersection of two extended spike-cones that belong

to different legs (of variable patterns), add a guard and move these guards to the corresponding points v_1 or v_5 in the variable patterns. Intersections of three extended spike-cones that belong to three different legs of variable patterns do not exist.
– Guards at other points can be moved to any point v_1 or v_5 of any variable pattern, if there is no guard at v_1 or v_5 already. Guards in intersections of two spike-cones of the same variable pattern are moved to point v_1 or v_5 of the variable pattern, if there is no guard there already.

The solution obtained by moving and adding guards as indicated is still feasible, i.e., the guards still see all of the polygon.

We are now ready to set the truth values of the variables. For each variable pattern x_j, if there is a guard at point v_5 and no guard at point v_1, let x_j be false. If there is a guard at point v_1 and no guard at point v_5, let x_j be true. If there is a guard at both v_1 and v_5, then set x_j in such a way that a majority of the literals of x_j become true.

4 Analysis of the Reduction

Consider the promise problem of 5-Occurrence-3-Sat, where we are given an instance of 5-Occurrence-3-Sat, and we are promised that the instance is either satisfiable or at most $m(1-4\epsilon)$ clauses are satisfiable by any assignment of the variables. The NP-hardness of this problem for small enough values of ϵ follows from the fact that 5-Occurrence-3-Sat is $MAXSNP$-complete (see [12] and [2]). An analysis of our reduction leads to the following.

Theorem 1. *Let I be an instance of the promise problem of* 5-Occurrence-3-Sat*, let n be the number of variables in I and let $m \leq \frac{5}{3}n$ be the number of clauses in I. Let $OPT(I)$ denote the maximum number of satisfiable clauses (for any assignment). Furthermore, let I' be the corresponding instance of PG and let $OPT(I')$ denote the minimum number of guards needed to completely see the polygon of I'. Then, the following hold:*

– *If $OPT(I) = m$, then $OPT(I') \leq 3m + n + 1$.*
– *If $OPT(I) \leq m(1-4\epsilon)$, then $OPT(I') \geq 3m + n + 1 + \epsilon m$.*

Theorem 1 shows that our reduction is gap-preserving (see [2]). It shows that the promise problem of PG with parameters $3m + n + 1$ and $3m + n + 1 + \epsilon m$ is NP-hard. Note that $m \geq \frac{n}{3}$, since each variable appears as a literal at least once. Therefore, unless $NP = P$, no polynomial time approximation algorithm for PG can achieve an approximation ratio of:

$$\frac{3m+n+1+\epsilon m}{3m+n+1} = 1 + \frac{\epsilon}{3+\frac{n+1}{m}} \geq 1 + \frac{\epsilon}{3+\frac{3(n+1)}{n}} \geq 1 + \frac{\epsilon}{7}$$

Thus, we have our main result:

Theorem 2. *The* POINT GUARD *problem for polygons without holes is APX-hard.*

Note that our proof works as well for the PG problem as defined in [8], where it is required that the guards see all of the interior and the boundary of the polygon.

5 Discussion

With similar arguments, we can show that the VERTEX GUARD and EDGE GUARD problems for polygon without holes are APX-hard as well. This characterization of the approximability of VERTEX GUARD, POINT GUARD and EDGE GUARD for polygons without holes is not the end of the story. We know of no approximation algorithm that achieves a constant ratio. We, therefore, did not focus on giving a concrete value for the inapproximability ratio. As long as no constant ratio approximation algorithms are known, it suffices to show that these problems are APX-hard. The approximation algorithms in [7] only achieve ratios of $O(\log n)$ for VG and EG.

References

1. P. Bose, T. Shermer, G. Toussaint and B. Zhu; *Guarding Polyhedral Terrains*; Computational Geometry 7, pp. 173-185, Elsevier Science B. V., 1997.
2. S. Arora, C. Lund; *Hardness of Approximations*; in: Approximation Algorithms for NP-Hard Problems (ed. Dorit Hochbaum), PWS Publishing Company, pp. 399-446, 1996. 428, 435
3. D. P. Bovet and P. Crescenzi; *Introduction to the Theory of Complexity*; Prentice Hall, 1993. 428
4. J. C. Culberson and R. A. Reckhow; *Covering Polygons is Hard*; Proc. 29th Symposium on Foundations of Computer Science, 1988. 427
5. St. Eidenbenz, Ch. Stamm, P. Widmayer; *Positioning Guards at Fixed Height above a Terrain - an Optimum Inapproximability Result*; to appear in Proceedings ESA, 1998. 433
6. St. Eidenbenz, Ch. Stamm, P. Widmayer; *Inapproximability of some Art Gallery Problems*; to appear in Proceedings CCCG, 1998. 428
7. S. Ghosh; *Approximation Algorithms for Art Gallery Problems*; Proc. of the Canadian Information Processing Society Congress, 1987. 427, 428, 436
8. D. T. Lee and A. K. Lin; *Computational Complexity of Art Gallery Problems*; in IEEE Trans. Info. Th, pp. 276-282, IT-32 (1986). 427, 428, 436
9. B. Nilsson; *Guarding Art Galleries - Methods for Mobile Guards*; doctoral thesis, Department of Computer Science, Lund University, 1994. 427
10. J. O'Rourke and K. J. Supowit; *Some NP-hard Polygon Decomposition Problems*; IEEE Transactions on Information Theory, Vol IT-29, No. 2, 1983. 427
11. J. O'Rourke; *Art Gallery Theorems and Algorithms*; Oxford University Press, New York (1987). 427
12. C.H. Papadimitriou and M. Yannakakis; *Optimization, Approximation, and Complexity Classes*; Proc. 20th ACM Symposium on the Theory of Computing, 1988. 428, 435

13. T. Shermer; *Recent Results in Art Galleries*; Proc. of the IEEE, 1992. 427
14. J. Urrutia; *Art Gallery and Illumination Problems*; to appear in Handbook on Computational Geometry, edited by J.-R. Sack and J. Urrutia, 1998. 427, 428

The Inapproximability of Non NP-hard Optimization Problems

Liming Cai[1], David Juedes[1], and Iyad Kanj[2]

[1] School of EE and CS, Ohio University
Athens, Ohio 45701, USA
[2] Department of Computer Science, Texas A&M University
College Station, Texas 77834, USA

Abstract. The inapproximability of non NP-hard optimization problems is investigated. Based on *self-reducibility* and approximation preserving reductions, it is shown that problems Log Dominating Set, Tournament Dominating Set and Rich Hypergraph Vertex Cover cannot be approximated to a constant ratio in polynomial time unless the corresponding NP-hard versions are also approximable in deterministic subexponential time. A direct connection is established between non NP-hard problems and a PCP characterization of NP. Reductions from the PCP characterization show that Log Clique is not approximable in polynomial time and Max Sparse SAT does not have a PTAS under the assumption that SAT cannot be solved in deterministic $2^{O(\log n \sqrt{n})}$ time and that NP $\not\subseteq$ DTIME$(2^{o(n)})$.

1 Introduction

Recent work on the approximability of NP-hard optimization problems has shown that the task of computing good approximate solutions for such problems is often as hard as computing exact solutions. (See [2,21].) Indeed, our ability to find even remotely good approximation algorithms (within n^ϵ of optimal) for problems such as minimal graph coloring hinges upon the central question in complexity theory, namely, whether or not P is equal to NP. The technical mechanisms that allow us to prove such strong results come from work on probabilistically checkable proofs and, most notably, from the probabilistic characterization of NP, the well-known PCP theorem [3,4]. It is natural to ask whether or not the approximability of non NP-hard problems can be determined using similar techniques.

Certain natural problems, such as Tournament Dominating Set and computing the Vapnis-Chervonenkis (V-C) Dimension, and many restricted versions of NP-complete problems have the property that they can be solved with a limited (i.e., $O(\log^k n)$) amount of nondeterminism, yet are not known to have efficient deterministic solutions. Such problems have been investigated in the literature [5,7,9,10,14,17,22,23,26]. (See [19] for a survey of results.) Since these problems can be solved deterministically in $O(n^{\log^k n})$ time, it is unlikely that these problems are NP-hard since this would imply that every problem in NP is computable in quasi-polynomial time.

In this extended abstract, we are particularly interested in the following problems.

> Max Sparse SAT: Given a set F of clauses, $C_1, \cdots, C_m$, where F contains at most $\log^2 m$ boolean variables, find an assignment to the variables that satisfies the maximum number of the clauses.
> Tournament Dominating Set: Given a tournament graph G, find a minimum dominating set for the graph.
> Log^k Dominating Set: $(k \geq 1)$ Given a directed graph $G = (V, E)$ in which $V = \{v_1, \ldots, v_n\}$ and subset $\{v_1, \cdots, v_{\log^k n}\}$ of V forms a dominating set for G, find a minimum dominating set for G. (We write Log Dominating Set for Log^1 Dominating Set.)
> Rich Hypergraph Vertex Cover: Given a hypergraph H with n vertices in which each hyperedge contains at least half of the vertices, find a minimum vertex cover for H.
> Log^k Hypergraph Vertex Cover: $(k \geq 1)$ Given a hypergraph $H = (V, E)$ in which $V = \{v_1, \ldots, v_n\}$ and subset $\{v_1, \cdots, v_{\log^k n}\}$ of V forms a vertex cover for H, find a minimum vertex cover for H.
> Log^k Clique: $(k \geq 1)$ Given a graph G of n vertices, find a maximum clique for G such that the clique size is bounded by $\log^k n$.

While it is unlikely that any of the above problems is NP-hard, it is also unlikely that any of these problems is computable in polynomial time. In particular, Papadimitriou and Yannakakis [23] show that decision versions of Log Dominating Set, Tournament Dominating Set, and Rich Hypergraph Vertex Cover are all complete for the syntactical class LOGSNP and that a decision version of V-C Dimension is complete for LOGNP. Working on fixed-parameter tractability, Downey and Fellows [12,13] prove that the parameterized versions of V-C Dimension and Tournament Dominating Set are $W[1]$-hard and $W[2]$-hard, respectively, under the uniform reduction. Cai and Chen [7] give an alternative hardness characterization of the complexity for these problems and demonstrate that $\Theta(\log^2 n)$ nondeterminism is essential to solving these problems. Feige and Kilian [17] prove that if Log Clique is computable in polynomial time, then $\mathrm{NTIME}(f(n)) \subseteq \mathrm{DTIME}(2^{O(\sqrt{f(n) polylog f(n)})})$. Megiddo and Vishkin [22] prove that there are polynomial-time algorithms for Tournament Dominating Set and Sparse SAT if and only if SAT can be computed in time $O(2^{\sqrt{v}} n^c)$ time, where v is the number of variables, n is the length of the formula and c is a constant.

As mentioned above, the best-known algorithms for this class of non NP-hard optimization problems run in time $O(n^{\log^k n})$. Since these algorithms perform poorly on reasonably large input instances (e.g., when $n = 500$ or larger), it is desirable to design efficient approximation algorithms for these problems. The limits of our ability to approximate these problems lies largely unexplored. However, there is evidence that these problems are hard to approximate. Recently, through the work on the probabilistic characterization of NP, Feige and

Kilian [17] show that Log Clique cannot be approximated to any constant ratio in polynomial time unless there is a subexponential randomized simulation of linear nondeterministic computation. Cai and Chen [8] show that it is unlikely that V-C Dimension has a full polynomial-time approximation scheme since such a scheme would imply the collapse of the W-hierarchy [11] and the consequence of SAT $\in$ DTIME$(2^{o(n)})$ [9].

We expect that more detailed approximability results for the above problems will reveal intrinsic natural properties of the underlying problems just as approximability research on NP-hard optimization problem revealed similar results. Because of the nature of the above problems, existing inapproximability results for NP-hard optimization problems cannot be directly applied. Our primary interest here is to build techniques to demonstrate the negative aspects of the approximation of these non NP-hard problems.

Many NP-hard optimization problems are *self-reducible* in the sense that they can be reduced to a non NP-hard version of the same problem (e.g., with solutions restricted to size $\log n$) in super-polynomial time, and their solutions can be related in a uniform manner. Feige and Kilian [17] use this approach to investigate the complexity of Log Clique. In Section 2, we obtain related results. We show that Hypergraph Vertex Cover and Dominating Set are both self-reducible. The fact enables us to show, in particular, that Log Dominating Set cannot be approximated in polynomial time to any constant ratio unless the corresponding NP-hard problem can be approximated to a constant ratio in time $O(2^{n^\delta})$, for some $\delta < 1$. Similar results are proved for Log Hypergraph Vertex Cover.

In Section 3, we prove inapproximability of non NP-hard problems based on PCP characterizations. First, we show that class PCP(r, q) is equal to class PCP$(cr, 2^{(1-c)r}q)$, for any $c < 1$, and that the latter can be reduced to Log^k Clique to obtain an inapproximability result for problem Log^k Clique, for some $k > 1$. Then we apply the technique to the PCP characterization of NP by Fotakis and Spirakis [18] which uses strictly $\log n + t$ random bits and a constant number of query bits for each instance x of length n, where t is independent of n. We reduce the number of random bits to $d \log n$, for $d < 1$, and thus increase the number of query bits to the factor of n^{1-d}. The latter characterization can be directly reduced to Max Sparse SAT and Log Clique. We show that Max Sparse SAT does not have a polynomial-time approximation scheme and that Log Clique cannot be approximated to any constant ratio in polynomial-time unless NP $\subseteq$ DTIME$(2^{O(\log n \sqrt{n})})$. The latter result improves the one by Feige and Kilian [17].

In Section 4, we build a number of nontrivial approximation-preserving reductions, making it possible to extend inapproximability results to more natural non NP-hard problems such as Rich Hypergraph Vertex Cover and Tournament Dominating Set. In particular, we show that for any $c \geq 1$, these two problems, along with several others, cannot be approximated to the ratio c in polynomial time unless NP-hard problem Dominating Set can be approximated to the ratio $2c$ in time $O(2^{n^\delta})$, for some $\delta < 1$.

All proofs are omitted from this extended abstract.

2 Self-Reducibility

For restricted versions of NP-hard problems, it is natural to ask whether the approximability of a problem becomes more manageable because of this restriction. The central question is whether approximability is uniformly distributed over all instances of a problem. In this section, we investigate the self-reducibility of optimization problems. Feige and Kilian [17] previously used this approach to study the complexity of Log Clique. We begin by introducing the notion of self-reducibility.

Definition 1. Let π be a maximization problem. The problem π is *self-reducible* if there is a function $f\colon \pi \to \pi$, and an unbounded function $t(n)$ such that

(1) for any instance I of π, with length n, and for any $k \geq 0$, there is a solution to instance $f(I)$ with value *at least* m if and only if there is a solution to instance I with value *at least* $m \cdot t(n)$;
(2) the function f is computable in time $O(2^{n^{\delta}})$, for some $\delta < 1$.

By replacing *at least* by *at most* in the above definition, we may similarly define the self-reducibility of minimization problems.

For integers $k \geq 1$, let $\text{Log}^k\pi$ be a restricted version of optimization problem π in which each restricted instance of size N has solution values bounded by $\log^k N$. It is not hard to see that, if π is self-reducible and all solution values are bounded by some polynomial, then π can be reduced to $\text{Log}^k\pi$ in time $O(2^{n^{\delta}})$.

Proposition 1. (Feige and Kilian [17]) Clique *is self-reducible* (*to problem* Log^k Clique*, for any* $k \geq 1$).

Here we extend this work to other problems.

Consider a given directed graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$, let $D_1, \ldots, D_m$ be all subsets of V of size at most n^c, for $c < 1$. Construct a new graph $H_G = (V', E')$, where $V' = V \cup U$, $U = \{u_0, u_1, \ldots, u_m\}$, and

$$E' = \{(u_0, u_i) : 1 \leq i \leq m\} \cup \{(u_i, v_j) : v_j \in D_i, \text{ or } \exists v \in D_i \text{ s.t. } (v, v_j) \in E\}$$

Based on this reduction, it is not hard to justify the following lemma.

Lemma 1. Dominating Set *is self-reducible* (*to* Log^k Dominating Set*, for any* $k \geq 1$).

Lemma 2. Hypergraph Vertex Cover *is self-reducible* (*to* Log^k Hypergraph Vertex Cover*, for any* $k \geq 1$).

In the proofs of the self-reducibility for above optimization problems, the constructed reductions from instances to restricted ones suggest that solutions for the latter can be used to obtain solutions for the former. This leads to the following connection between the inapproximability of restricted optimization problems and their NP-hard versions.

Theorem 1. *Let π be any of the problems* Dominating Set, Hypergraph Vertex Cover *and* Clique. *Then for some $k \geq 1$, the restricted optimization problem* Log$^k\pi$ *cannot be approximated in polynomial time to any constant ratio unless π can be approximated to some constant ratio in time $O(2^{n^\delta})$, for some $\delta < 1$.*

It is still an open question whether the NP-hard optimization problems Dominating Set, Hypergraph Vertex Cover, or Clique can be approximated to some constant ratio in time $O(2^{n^\delta})$, for some $\delta < 1$. However, it has been shown that [3,4,6,16] that they do not adopt polynomial-time approximation algorithms achieving any constant ratio unless P=NP. These results (even with worse ratios) have been obtained through the connection between recently developed probabilistic characterizations of NP languages and the inapproximability of these problems. Based on probabilistic checkable proofs, we are able to relate the inapproximability of Log^r Dominating Set, Log^k Hypergraph Vertex Cover and Log^r Clique to open questions in complexity theory.

For our purpose, we give a definition for PCP protocols and refer the reader to [2] for more details.

Definition 2. *A language L is in the class* PCP(r, q) *if there exists a randomized polynomial time verifier V such that for every input x (i) random strings generated by V are of length r, and the number of bits in a proof (witness) probed by V is at most q, (ii) if $x \in L$, there is a proof such that V accepts x for every generated random string, and (iii) if $x \notin L$, for any proof, V rejects x for at least half of the generated random strings.*

Proposition 2. (PCP Theorem [3,4]) NP = PCP$(O(\log n), O(1))$.

Theorem 2. *Let π be any of the problems* Dominating Set, Hypergraph Vertex Cover *and* Clique. *Then for some $k \geq 1$,* Log$^k\pi$ *cannot be approximated to any constant ratio unless* SAT *can be solved in deterministic time $O(2^{n^\delta})$, for some $\delta < 1$.*

3 PCP and Inapproximability

An alternate PCP characterization of NP can be used to establish a direct connection between PCP and the inapproximability of non-NP-hard optimization problems. Our aim is to strengthen the inapproximability results given in the previous section. For this, we need to reduce the number of random bits used by the verifier. First, we observe that the number of random bits can be reduced by introducing more queries [1].

Lemma 3. *Let r and q be two constants. A language L is in* PCP$(r \log n, q)$, *then it is in* PCP$(cr \log n,\ qn^{(1-c)r})$ *for any constant c, $0 < c < 1$.*

Lemma 3 enables us to present an alternative proof for Theorem 2 (in particular, for Log^k Clique without resorting to using the self-reducibility of the problem). The key is that a PCP($cr \log n$, $qn^{(1-c)r}$) characterization of an NP language can be reduced to LOG^r CLIQUE in time $O(2^{o(n)})$. The reduction is an extension of the generic one used in [16].

Theorem 2 does not apply to problems LOG^k π, for $k < (r-\delta)/\delta$. This is because that k is determined by the constant factor r for the logarithmic function that bounds the number of random bits used by verifiers. In particular, $k \geq r$ when $\delta = r/(r+1)$.

In order to show the same inapproximability for problems LOG^k π, for $k < r$, we need to further scale down the number of random bits, while at the same time keeping the number of query bits small. For this, we base our techniques on the new PCP characterization of NP languages introduced by Fotakis and Spirakis [18] which uses strictly $\log n + t$ random bits, where t is independent of n, and a constant number of query bits for each instance x of length n. Moreover, by Lemma 3, we obtain

Theorem 3. *For any $d, e < 1$ with $e + d \geq 1$, every NP language admits a* PCP($d \log n + t, qn^e$) *characterization, for some q, t independent of n.*

Corollary 1. *Every NP language admits a* PCP($\frac{1}{2} \log n + t, q\sqrt{n}$) *characterization, for some q, t independent of n.*

The new PCP characterizations given in Theorem 3 and Corollary 1 allow us to improve some results given in the previous section.

Theorem 4. LOG CLIQUE *cannot be approximated to any constant ratio in polynomial time unless* SAT *can be solved in deterministic time* $2^{O(\log n \sqrt{n})}$.

Corollary 2. *For any constant $k \geq 1$,* LOG^k CLIQUE *cannot be approximated to any constant ratio in polynomial time unless NP $\subseteq$ DTIME(2^{n^δ}), for some fixed $\delta < 1$.*

Theorem 4 and its corollary improve the result by Feige and Killian [17] that LOG CLIQUE cannot be approximated to any constant ratio unless there is a subexponential randomized simulation of nondeterministic computation.

The new PCP characterization also leads to inapproximability results for MAX SPARSE SAT. By extending the generic reduction from PCP to MAX 3SAT in [2], we can prove that

Theorem 5. *Problem* MAX SPARSE SAT *does not have polynomial time approximation scheme unless* SAT *can be solved in time* $2^{O(\log n \sqrt{n})}$.

Corollary 3. *Problem* MAX SPARSE SAT *does not have polynomial time approximation scheme unless NP $\subseteq$ DTIME(2^{n^δ}), for some fixed $\delta < 1$.*

Note that Theorem 5 and the corollary complement the result that MAX 3SAT does not have polynomial time approximation scheme unless P=NP [3]. The consequence that all NP languages can be computed in time $O(2^{o(n)})$ does not seem strong as P=NP; however, it is widely considered to be unlikely.

Corollary 4. *For some $k \geq 1$,* LOG^k DOMINATING SET *cannot be approximated to any constant ratio in polynomial time unless* NP $\subseteq$ DTIME(2^{n^δ})*, for some fixed $\delta < 1$.*

Corollary 5. *For some $k \geq 1$,* LOG^k HYPERGRAPH VERTEX COVER *cannot be approximated to any constant ratio in polynomial time unless* NP $\subseteq$ DTIME(2^{n^δ})*, for some fixed $\delta < 1$.*

4 Approximability-Preserving Reductions

Approximability-preserving reductions have proved effective in demonstrating inapproximability of NP-hard optimization problems. The nature of non-NP-hard optimization problems shows that it is harder to devise approximability-preserving reductions among them because any such reduction should preserve not only approximability but also *restrictions* enforced on these problems.

In this section, we present a number of approximability-preserving reductions to show that problems LOG DOMINATING SET, TOURNAMENT DOMINATING SET and RICH HYPERGRAPH VERTEX COVER, equivalent in complexity [23], are also equivalent in approximability.

First we construct a reduction from RICH HYPERGRAPH VERTEX COVER to TOURNAMENT DOMINATING SET.

Given a hypergraph $H = (V, E)$, where hyperedges $E = \{e_0, \cdots, e_{m-1}\}$ and $V = \{v_1, \cdots, v_n\}$, we construct a tournament graph $T = (V_T, E_T)$ such that $V_T = V \cup U$, for $V = \{v_1, \cdots, v_n\}$ and

$$U = \{u_j^i \,|\, 0 \leq i \leq p-1, e_j \in E\} \cup \{u_j^i \,|\, 0 \leq i \leq p-1, m \leq j \leq p-1\}$$

where $p \geq m$ is a prime to be defined later; and $E_T = \bigcup_{i=1}^{6} E_i$, where

$$\begin{aligned}
&E_1 = \{(v_s, v_t) | 1 \leq s < t \leq n\}, \quad E_2 = \textstyle\bigcup_{j=0}^{p-1}\{(u_j^i, u_j^k) | 0 \leq i < k \leq p-1\}\\
&E_3 = \{(u_s^i, u_t^k) | s - t \equiv a^2 \,(mod\, p) \text{ for some } a \in Z_p,\ 0 \leq s, t \leq p-1,\\
&\qquad \text{and } 0 \leq i, k \leq p-1\}\\
&E_4 = \{(v_s, u) | u \in G_j \text{ and } v_s \in e_j,\ 1 \leq s \leq n, \text{ and } 0 \leq j \leq m-1\}\\
&E_5 = \{(v_s, u) | u \in G_j,\ 1 \leq s \leq n, \text{ and } m \leq j \leq p-1\},\\
&E_6 = \{(u, v) | u \in U,\ v \in V, \text{ and } (v, u) \notin \textstyle\bigcup_{h=1}^{5} E_h\}
\end{aligned}$$

where for each $j = 0, \cdots, p-1$, set G_j is defined as $G_j = \{u_k^i \,|\, k = i + j(mod\, p),\ i = 0, 1, \cdots, p-1\}$.

Intuitively, the tournament T is so constructed that it contains $n+p^2$ vertices that is composed of n vertices of $V = \{v_1, \cdots, v_n\}$ of H and p copies of p items $u_0, \cdots, u_p$ (the first m item corresponding the m hyperedges in H and the last $p-m$ items being dummies). Let $U = \{u_j^i | 0 \leq i, j \leq p-1\}$. The p^2 vertices in U form a $p \times p$ matrix layout. Column j is a set of p copies of u_j and row i is the ith copy of $\{u_0, \cdots, u_{p-1}\}$.

Moreover, the edges of T are constructed in the following way. For vertices in V, vertices with a lower index dominate those with a higher one. Consider within matrix U, for each column, vertices with a lower superscript dominate those with a higher one. For each i, within row i, vertices are connected according to a construction proposed by Meggido and Vishkin [22] where vertex u_s^i dominates vertex u_t^i if and only if $s-t$ is a square in the field Z_p. They show that $s-t$ is a square if and only if $t-s$ is not a square. This guarantees a construction of tournament edges for each row. Moreover, such construction is extended in U to vertices belonging to any two different rows. I.e., vertex u_s^i dominates vertex u_t^k if and only if $s-t$ is a square in the field Z_p, $i, k = 0, \cdots, p-1$.

The edge construction between vertices in V and vertices in U can be explained as follows. Logically, matrix U is diagonally partitioned into p sets $G_0, \cdots, G_{p-1}$, where $G_j = \{u_k^i \,|\, k = i + j (mod\, p),\ i = 0, 1, \cdots, p-1\}$. Informally, each set G_j contains one vertex from each row and one from each column. Specifically, $G_0 = \{u_0^0, u_1^1, \cdots, u_{p-1}^{p-1}\}$, $G_1 = \{u_1^0, u_2^1, \cdots, u_{p-1}^{p-2}, u_0^{p-1}\}$, $\cdots$, $G_{p-1} = \{u_{p-1}^0, u_0^1, \cdots, u_{p-2}^{p-1}\}$. Then $G_j \cap G_i = \phi$ for $i \neq j$. Also for each j, the set G_j contains vertex u_j^0 from the first row.

For each vertex $v_i \in V$, if $v_i \in e_j$ in the hypergraph H, then for each $u \in G_j$, there is an (directed) edge (v_i, u) in T. In other word, in the tournament T, v_i dominates the vertex u_j^0 in the first row of the matrix and all other vertices in the set G_j if and only if a vertex v_i covers hyperedge e_j in the hypergraph H. After this, for each $v \in V$ and $u \in U$, add (u, v) to E_T if (v, u) is not already in E_T.

Now we identify the prime number p used in the construction. Let q be such that $m \leq 2^{2q} q^2$. p is chosen so that $p > 2^{2q} q^2$ and $p \equiv 3 (mod\, 4)$. Note that q can be chosen to be the smallest number that is at least $\log m$. By Proposition 2.11 of Meggido and Vishkin [22], the growth rate of the function $p^*(q)$ that assigns a prime p to each number q as above is $O(2^{2q} q^2)$.

Lemma 4. *For the constructed tournament T, let $V' \subseteq V$ and $U' \subseteq U$. If $V' \cup U'$ is a dominating set for T, then either there is a subset of U of size $|U'| + 1$ dominating T or there is a subset of V of size $|V'| + 1$ dominating T.*

Proposition 3. *Let $U_0 = \{u_j^0 \,|\, 0 \leq j \leq p-1\}$ in the matrix U. It needs at least q vertices from U_0 to dominate the whole set U_0, where q is the largest number such that $p > 2^{2q} q^2$.*

Lemma 5. *Let D be a dominating set for the tournament T constructed in the reduction. Then there is subset of V of size at most $|D|$ that dominates T.*

Suppose that there is an approximation algorithm A that finds an approximated dominating set D for each tournament T constructed and that achieves ratio c for some $c \geq 1$. By Lemma 5 and its proof, there is a dominating set V' for T, $V' \subseteq V$ and $|V'| \leq |D|$. It is easy to see that vertices in V' cover all hyperedges in H. Hence, the approximated vertex cover size is bounded by $A(T)$. Moreover, $OPT(H) \geq OPT(T) - 1$ by edge construction of E_4 (assuming $OPT(T) > 1$). Then the ratio of approximating the hyperedge cover is bounded by $A(T)/(OPT(T) - 1) = c/(1 - 1/A(T)) \leq 2c$.

Note that all the work in the reduction can be done in polynomial time.

Theorem 6. *For any $c \geq 1$, in polynomial time,* Tournament Dominating Set *cannot be approximated to the ratio c unless* Rich Hypergraph Cover *can be approximated to the ratio $2c$.*

Theorem 7. *For each constant $r \geq 1$, in polynomial time, problem* Rich Hypergraph Vertex Cover *cannot be approximated to the ratio r unless problem* Log Dominating Set *can be approximated to the ratio r.*

Corollary 6. *For each constant $r \geq 1$, in polynomial time, problem* Log Hypergraph Vertex Cover *cannot be approximated to the ratio r unless problem* Log Dominating Set *can be approximated to the ratio r.*

Theorem 8. *For any $c \geq 1$,* Log Dominating Set *cannot be approximated in polynomial time to the ratio c unless* Tournament Dominating Set *can be approximated to the same ratio.*

Corollary 7. *For any constant $c > 1$,* Tournament Dominating Set*,* Log Dominating Set*,* Rich Hypergraph Vertex Cover*, and* Log Hypergraph Vertex Cover *cannot be approximated to the ratio c in polynomial time unless* Dominating Set *can be approximated to the ratio $2c$ in time $O(2^{n^\delta})$, for some $\delta < 1$.*

References

1. Arora, S.: Personal Communication. 441
2. Arora, S., Lund, C.: Hardness of Approximations. In Approximation Algorithms for NP-hard problems. Ed. Dorit Hochbaum. (1997). 437, 441, 442
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Verification and hardness of approximation problems. Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science. (1992) 14–23. 437, 441, 442
4. Arora, S., Safra, S.: Probabilistic Checking of Proofs. Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science. (1992) 2–13. 437, 441
5. Buss, J. F., Goldsmith, J.: Nondeterminism within P. SIAM Journal on Computing. **22** (1993) 560–572. 437
6. Bellare, M., Goldwasser, S., Lund, C., Russell, A.: Efficient probabilistic checking of proofs and applications to approximation. Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (1993) 113–131. 441
7. Cai, L., Chen, J.: On the amount of nondeterminism and the power of verifying. SIAM Journal on Computing. **26** (1997) 733–750. 437, 438
8. Cai, L., Chen, J.: On Fixed-Parameter Tractability and Approximability of NP Optimization Problems. Journal of Computer and System Sciences. **54** (1997) 465–474. 439
9. Cai, L., Chen, J., Downey, R., Fellows, M.: On the Structure of Parameterized Problems in NP. Information and Computation. **123** (1995) 38–49. 437, 439
10. Díaz, J., Torán, J.: Classes of bounded nondeterminism. Math. System Theory. **23** (1990) 21-32. 437
11. Downey, R., Fellows, M.: Fixed-Parameter Intractability. Proceedings of the 7th IEEE Annual Conference on Structure in Complexity Theory. (1992) 36–49. 439

12. Downey, R., Fellows, M.: Fixed-Parameter Tractability and Completeness I: Basic Results. SIAM Journal on Computing. **24** (1995) 873–921. 438
13. Downey, R., Fellows, M.: Parameterized Computational Feasibility. The Proceedings of FEASMATH: Feasible Mathematics: A Mathematical Sciences Institute Workshop (1995). 438
14. Farr, G.: On problems with short certificates. Acta Informatica. **31** (1994) 479–502. 437
15. Feige, U.: A Threshold of $\ln n$ for Approximating Set Cover. Proceedings of The 28th Annual ACM Symposium On The Theory Of Computing. (1996) 314–318.
16. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy M.: Approximating clique is almost NP-complete. Proceedings of the 32nd IEEE Symposium on the Foundations of Computer Science. (1991) 2–12. 441, 442
17. Feige, U., Kilian, J.: On Limited versus Polynomial Nondeterminism. CJTCS: Chicago Journal of Theoretical Computer Science. (1997). 437, 438, 439, 440, 442
18. Fotakis, D., Spirakis, P.: (poly(loglogn), poly(loglogn))-Restricted Verifiers are Unlikely to exist for languages in NP. Proceedings of the 23nd International Symposium on Mathematical Foundations of Computer Science. (1996) 360-371. 439, 442
19. Goldsmith J., Levy, M., Mundhenk, M.: Limited Nondeterminism. SIGACT News. **27** (1996) 20–29. 437
20. Johnson, D. S.: The NP-Completeness Column: Ongoing Guide. Journal of Algorithms. **13** (1992) 502–524.
21. Kann, V.: On the Approximability of NP-complete Optimization Problems. Royal Institute of Technology, Sweden. Ph.D. Thesis (1992). 437
22. Megiddo, N., Vishkin, U.: On Finding a Minimum Dominating Set in a Tournament. Theoretical Computer Science. **61** (1988) 307–316. 437, 438, 444
23. Papadimitriou C. H., Yannakakis M.: On Limited Nondeterminism and the Complexity of the V-C Dimension. Proceedings of the 8th Annual Conference on Structure in Complexity Theory. (1993) 12–18. 437, 438, 443
24. Pippenger, N., Fischer, M. J.: Relations Among Complexity Measures. Journal of the ACM. **26** (1979) 361–381.
25. Polishchuk, A., Spielman, D. A.: Nearly-linear Size Holographic Proofs. Proceedings of the 26th Annual ACM Symposium on the Theory of Computing. (1994) 194–203.
26. Szelepcsényi, R.: β_k-complete problems and greediness. Computer Science Department, University, Rochester, New York. Technical Report **445** (1993). 437

An Efficient $\mathcal{NC}$ Algorithm for a Sparse k-Edge-Connectivity Certificate*

Hiroshi Nagamochi and Toru Hasunuma

Department of Applied Mathematics and Physics
Graduate School of Informatics, Kyoto University
{naga,hasunuma}@kuamp.kyoto-u.ac.jp

Abstract. We present an efficient $\mathcal{NC}$ algorithm for finding a sparse k-edge-connectivity certificate of a multigraph G. Our algorithm runs in $O((\log kn)(\log k)^2(\log n)^2)$ time using $O(k(n + m'))$ processors on an ARBITRARY CRCW PRAM, where n and m' stand for the numbers of vertices in G and edges in the simplified graph of G, respectively.

1 Introduction

In this paper, we treat multigraphs without loops. Let G be a multigraph. Let $V(G)$ and $E(G)$ denote the vertex set and the edge set of G, respectively. Also, let $n = |V(G)|$ and $m = |E(G)|$. A *k-edge-connectivity certificate* (resp., *k-vertex-connectivity certificate*) of G is a spanning submultigraph H of G such that for any two vertices u, v and any positive integer $k' \leq k$, there are k' edge-disjoint (resp., internally disjoint) paths between u and v in H if and only if there are k' edge-disjoint (resp., internally disjoint) paths between u and v in G. We say that a k-edge-connectivity certificate H is *sparse* if $|E(H)| = O(kn)$. (Similarly, a sparse k-vertex-connectivity certificate is defined.) Computing a sparse k-edge-connectivity (k-vertex-connectivity) certificate is useful to several graph connectivity problems ([5],[9]), since we can reduce the size of a multigraph while preserving its k-connectedness.

Nagamochi and Ibaraki [8] gave a linear-time sequential algorithm for finding a sparse k-edge-connectivity (and also k-vertex-connectivity certificate). In parallel setting, Cheriyan *et al.* [1] presented algorithms that compute such certificates in $O(k \log n)$ time using $C(n, m)$ processors, where $C(n, m)$ is the number of processors required to compute a spanning tree in each connected component in $O(\log n)$ time. So far, no $\mathcal{NC}$ algorithm for a sparse k-vertex-connectivity certificate is known.

On the other hand, Karger and Motwani [5] gave the first $\mathcal{NC}$ algorithm for finding a sparse k-edge-connectivity certificate as one building block to construct an $\mathcal{NC}$ algorithm for the minimum cut problem. Their algorithm runs

* This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan and the subsidy from the Inamori Foundation.

in $O((\log m)(\log(kn + m))^3)$ time using $O(km)$ processors on an ARBITRARY CRCW PRAM (although the time bound is not explicitly written in their paper).

In this paper, we present an efficient $\mathcal{NC}$ algorithm for a sparse k-edge-connectivity certificate. Our algorithm runs in $O((\log kn)(\log k)^2(\log n)^2)$ time using $O(k(n+m'))$ processors on an ARBITRARY CRCW PRAM, where $m'(= O(n^2))$ is the number of edges in the simplified graph of G, and the input multigraph G is assumed to be represented by the simplified graph with a multiplicity function. Clearly, $m' \leq m$, and m may not be bounded by n^2 in a multigraph. Thus, our algorithm considerably improves the number of processors in a multigraph having many multiple edges.

Karger and Motwani also showed that a $(2 + \epsilon)$-approximate minimum cut can be found in $\mathcal{NC}$ using $O(m^2/n)$ processors. Adapting our algorithm to their approximate minimum cut algorithm, the number of processors to compute a $(2 + \epsilon)$-approximate minimum cut in $\mathcal{NC}$ can be reduced to $O(nm)$ processors.

This paper is organized as follows. In Section 2, several basic definitions and results used in the paper are given. In Section 3, we give a simple algorithm for a sparse k-edge-connectivity certificate. The algorithm consists of $O(\log n)$ phases in which we need to compute a maximal matching in a bipartite graph. In Section 4, we analyze the time bound of Israeli and Shiloach's maximal matching algorithm in the case where an input graph is bipartite. In Section 5, we improve the number of processors of our algorithm for a sparse k-edge-connectivity certificate by using a scaling technique on multiplicity of a given multigraph.

2 Preliminaries

Let $G = (V, E, \phi_G)$ be a multigraph, where V is the vertex set, E is the edge set, and ϕ_G is a function from E to the set of unordered pair of distinct vertices. If $\phi_G(e) = \{u, v\}$, then we say that e is incident to u, u is incident to e, and e joins u and v. A multiedge $\langle u, v\rangle$ stands for the set of edges mapped to $\{u, v\}$ by ϕ_G, i.e., $\{e \in E(G) \mid \phi(e) = \{u, v\}\}$. The number of edges mapped to $\{u, v\}$ by ϕ_G is called the multiplicity of $\langle u, v\rangle$ and denoted by $\rho_G(\langle u, v\rangle)$. If $\rho_G(\langle u, v\rangle) = 1$ for any multiedge $\langle u, v\rangle$ of G, then G is a simple graph. The simplified graph of $S(G) = (V, E')$ is the simple graph obtained from G by replacing each multiedge with a single edge with the same end-vertices. In Section 5, we are given a multigraph G as a simple graph with multiplicity ρ_G (note that given a multigraph $G = (V, E, \phi_G)$, such a simplified graph can be obtained in $O(\log m)$ time using $O(m)$ processors).

Let $v \in V(G)$. The degree of v is denoted by $d(v)$. The maximum (resp., minimum) degree of vertices in G is denoted by $\Delta(G)$ (resp., $\delta(G)$). Let $S \subseteq V(G)$. If every edge $e \in E(G)$ is incident to at least one vertex in S, then S is called a *vertex cover* of G. A subset $M \subseteq E(G)$ is called a *matching* of G if any two edges in M are not incident to the same vertex. A *circuit* is an alternate closed sequence of vertices and edges, $v_0, e_1, v_1, \ldots, e_t, v_0$, where e_i joins v_{i-1} and $v_{i(mod\ t)}$, such that all edges used in the sequence are distinct. A circuit $v_0, e_1, v_1, \ldots, e_t, v_0$ is called a *cycle* if $v_i \neq v_j$ for $0 \leq i < j < t$. An *Euler* *circuit* of G is a circuit which passes through all edges of G. A *forest* is an acyclic simple graph. Let G_1 and G_2 be a submultigraphs of G. Then $G_1 \oplus G_2$ stands for the multigraph with $V = V(G_1) \cup V(G_2)$ and $E = E(G_1) \cup E(G_2)$. Also, $G_1 - G_2$ stands for the multigraph with $V = V(G_1)$ and $E = E(G_1) - E(G_2)$.

For a nonempty and proper subset $X \subseteq V(G)$, the set of edges that joins a vertex of X and a vertex of $\bar{X} = V - X$ is denoted by $E_G(X, \bar{X})$ and called a *cut* of G. The minimum size of cuts of G is called the *edge-connectivity* of G. If the edge-connectivity of G is at least k, then G is called k-edge-connected.

It is not difficult to see that a k-edge-connectivity certificate H of G can be also defined as a spanning submultigraph H of G such that

$$|E_H(X, \bar{X})| \geq \min\{k, |E_G(X, \bar{X})|\} \tag{1}$$

for all nonempty subset $X \subset V(G)$.

One simple way to obtain a sparse k-edge-connectivity certificate of G, which is given by [8], is as follows. First, construct a maximal forest F_1 of G. Next, construct a maximal forest F_2 of $G - F_1$. Iterate this process, i.e., construct a maximal forest F_i of $G - \oplus_{1 \leq j < i} F_j$, until we obtain F_k. Then it is easy to see that $H = \oplus_{1 \leq i \leq k} F_i$ satisfies (1) and $|E(H)| \leq k(n-1)$. Thus $H = \oplus_{1 \leq i \leq k} F_i$ is a sparse k-edge-connectivity certificate of G.

Karger and Motwani observed this construction from the parallel point of view and defined a maximal k-jungle.

Definition 1. [5] *A k-jungle $\mathcal{J}$ of G is a set of k edge-disjoint forests of G. A k-jungle of G is maximal if for every forest $F \in \mathcal{J}$, adding any edge e not used in any forest in the jungle to F creates a cycle.* □

Let $\mathcal{J} = \{F_1, \ldots, F_k\}$ be a k-jungle of G. Although $\mathcal{J}$ is defined as a set of forests, we often consider $\mathcal{J}$ as the multigraph $\oplus_{1 \leq i \leq k} F_i$. Also, $\cup_{F \in \mathcal{J}} E(F)$ is abbreviated to $E(\mathcal{J})$. A maximal k-jungle of G is a sparse k-edge-connectivity certificate of G. Similarly to Karger and Motwani's algorithm, our algorithm for a sparse k-edge-connectivity certificate finds a maximal k-jungle of G.

3 A Simple Algorithm for a Maximal k-Jungle

Karger and Motwani proved that a maximal k-jungle can be obtained by solving the problem of finding a maximal matching in a bipartite graph $O(\log m)$ times. Since there is an $\mathcal{NC}$ algorithm for computing a maximal matching in a graph, their algorithm for computing maximal k-jungle is $\mathcal{NC}$. In this section, we give a simple algorithm that finds a maximal k-jungle by solving the maximal matching problem in a bipartite graph at most $O(\log n)$ times. Our algorithm is a natural extension of an $\mathcal{NC}$ algorithm for computing a maximal forest in a graph.

We use a well-known parallel algorithm ([10]) for computing connected components of a graph in $O(\log n)$ time. The algorithm can be easily modified to an algorithm for constructing a maximal forest of a graph. We prepare an array D on $V(G) = \{1, 2, \ldots, n\}$ to maintain the components of a forest F, where D is initially set by $D[v] := v$ for $v \in V(G)$. An array D defines a set of rooted trees by considering that $D[v] = u$ implies that u is a parent of v in a rooted tree ($D[v] = v$ if and only if v is the root of the tree containing v). (See standard books [2],[4],[6] for further details.)

To construct a maximum k-jungle, we prepare k arrays $D_1, \ldots, D_k$ for k edge-disjoint forests $F_1, \ldots, F_k$ (i.e., a k-jungle $\mathcal{J}$). Using these arrays D_p, $p = 1, \ldots, k$, we try to augment each forest F_p simultaneously by adding an edge which is not used in any forest in the current k-jungle $\mathcal{J}$. However, we need to avoid adding the same edge to more than one forest in $\mathcal{J}$. Such tie breaking can be done by solving the maximal matching problem in a certain bipartite graph.

Our algorithm consists of $O(\log n)$ executions of the following phase, called JUNGLE. Let $P = \{1, \ldots, k\}$. During the algorithm, we update a membership function $Q : E(G) \to P \cup \{0\}$ such that $Q[e] = p(> 0)$ implies that e has been used in the p-th forest F_p in the current k-jungle $\mathcal{J}$; $Q[e] = 0$ implies that e has not been used in any F_p in $\mathcal{J}$. Initially set $D_p[v] := v$ for all $v \in V$ and $p \in P$, and $Q[e] := 0$ for all $e \in E(G)$. A grafting operation merges some rooted trees defined by D_p into a fewer number of rooted trees. Given a membership function Q and a set of k arrays $D_1, \ldots, D_k$ on $V(G)$, a phase JUNGLE perform grafting operations and update Q and $\{D_p \mid p = 1, \ldots, k\}$ accordingly. The entire phase is described as follows, where $v_1(e)$ and $v_2(e)$ stand for the end vertices of e, and $STAR_p[v]$ is a function which returns TRUE if the rooted tree defined by D_p containing a vertex v is a rooted star. Exceptionally, we need to execute the conditional grafting two times consecutively at the first application of JUNGLE in order to guarantee the acyclicity of forests after the star grafting.

Procedure JUNGLE

```
1 . (Conditional grafting)
1.0. Let $EB := \emptyset$.
1.1. For all $(e,p) \in E(G) \times P$ pardo
  if $Q[e] = 0$ then begin
    if $STAR_p[v_1(e)]$ and $(D_p[v_2(e)] < D_p[v_1(e)])$ then add $((D_p[v_1(e)], p), e)$ to $EB$.
    if $STAR_p[v_2(e)]$ and $(D_p[v_1(e)] < D_p[v_2(e)])$ then add $((D_p[v_2(e)], p), e)$ to $EB$.
  end.
1.2. Let $B$ be the bipartite graph with $V(B) = (V(G) \times P) \cup E(G)$ and $E(B) = EB$.
1.3. Compute a maximal matching $M$ in $B$.
1.4. For all $((v,p), e) \in M$ pardo begin
  $Q[e] := p$.
  if $(D_p[v_2(e)] < D_p[v_1(e)])$ then $D_p[v] := D_p[v_2(e)]$, else $D_p[v] := D_p[v_1(e)]$.
  end.
2. (Star grafting)
2.0. Let $EB := \emptyset$.
2.1.For all $(e,p) \in E(G) \times P$ pardo
  if $Q[e] = 0$ then begin
    if $STAR_p[v_1(e)]$ and $(D_p[v_2(e)] \neq D_p[v_1(e)])$ then add $((D_p[v_1(e)], p), e)$ to $EB$.
    if $STAR_p[v_2(e)]$ and $(D_p[v_1(e)] \neq D_p[v_2(e)])$ then add $((D_p[v_2(e)], p), e)$ to $EB$.
  end.
```

```
2.2. Same as 1.2.
2.3. Same as 1.3.
2.4. For all $((v,p),e) \in M$ pardo begin
  $Q[e] := p$.
  if $STAR_p[v_1(e)]$ then $D_p[v] := D_p[v_2(e)]$, else $D_p[v] := D_p[v_1(e)]$.
  end.
3. (Pointer jumping)
3.1. If for all $(v,p) \in V(G) \times P$, $STAR_p[v]$ then EXIT.
3.2. For all $(v,p) \in V(G) \times P$, let $D_p[v] := D_p[D_p[v]]$.
```

Based on the similar analysis of the algorithm for a maximal forest and the fact that we compute a maximal matching in a bipartite graph in JUNGLE, we show that a maximal k-jungle is obtained by applying JUNGLE $O(\log n)$ times.

We see that each F_p is augmented in the same manner of the algorithm for computing a maximal forest. Thus, it is guaranteed that there is no cycle in each F_p in the resulting jungle. Also, since we compute a matching in steps 1.3 and 2.3 of JUNGLE, any two resulting forests do not have common edges. This preserves the edge-disjointness of forests in the resulting k-jungle. After a star grafting, if there is a rooted star defined by D_p, then the corresponding tree in the forest F_p cannot be augmented any more because we execute grafting operations based on "maximal" matching. Any grafting operation never increases the sum of the depths of all rooted trees defined by D_p. By the pointer jumping step, the depth of each rooted trees which is not a rooted star decreases by at least a factor $\frac{2}{3}$. Thus, after one application of JUNGLE, the sum of the depths of all rooted trees except for non-grafted rooted stars always decreases by a factor $\frac{2}{3}$. Thus, the following lemma holds.

Lemma 1. *A maximal k-jungle of G can be found by applying JUNGLE $O(\log n)$ times.* □

The number of processors required for JUNGLE is bounded by $O(k(n+m))$ if we can compute the maximal matching algorithm in steps 1.3 an 2.3 of JUNGLE using $O(k(n+m))$ processors. The time bound of JUNGLE depends on that of computing a maximal matching in a bipartite graph.

4 Maximal Matching in Bipartite Graphs

Israeli and Shiloach [3] gave an $\mathcal{NC}$ algorithm that computes a maximal matching in $O((\log n)^3)$ time using $O(n+m)$ processors on an ARBITRARY CRCW PRAM. In this section, we slightly modify Israeli and Shiloach's algorithm in the case of a bipartite graph so that it runs faster using its bipartiteness.

Let us first review their algorithm briefly. The top level description of the algorithm is written as follows. (To be precise, some modification is needed in the last few iterations of the inner while-loop [3, Theorem 3.4]. We, however, omit this since it does not directly affect the following argument.)

Let us call an iteration of the outer while-loop a phase. In a phase, given a graph G_i, a matching is computed by the inner while-loop, in which the degree of each vertex in G_i is gradually reduced to 0 or 1 (i.e., the resulting graph provides a matching).

Algorithm MAXIMAL-MATCHING

```
begin
  $M := \emptyset$; $i := 1$;
  While $E(G) \neq \emptyset$ do
     begin
       $G_i := G$;
       While $\Delta(G_i) > 1$ do DEGREE-SPLIT($G_i$);
       $M := M \cup E(G_i)$; { $E(G_i)$ is a matching in $G$ }
       $G := G-$ (all matched vertices in $G_i$);
       $i := i + 1$;
     end;
end.
```

Let l be the minimal integer satisfying $2^l \leq \Delta(G) \leq 2^{l+1} + 1$, i.e., $l = \lfloor \log(\Delta(G) - 2) \rfloor$. A vertex v satisfying $d(v) \geq 2^l$ is called *active.*

Procedure DEGREE-SPLIT(G)

1. Find all active vertices of G.
2. Construct the graph G_a induced by all edges of G incident with at least one active vertex.
3. Make G_a a graph of even degrees by connecting all the odd-degree vertices to a dummy vertex v^*.
4. Find an Euler circuit in every connected component of G_a.
5. In each connected component, label the edges of G_a by 0 and 1 alternately following the Euler circuit. In the component containing v^*, start your tour from it, labeling the first edge by 0. In the other components, start from any active vertex and label the first edge by 1.
6. Let G be the graph obtained from G by removing all the 0-labeled edges from G.

By applying DEGREE-SPLIT one time, the degrees of all active vertices of G_i become almost half, so does the maximum degree of G_i. Thus, after applying DEGREE-SPLIT $O(\log \Delta(G_i))$ times, the maximum degree of G_i becomes one, i.e., a matching of G_i is constructed. Then, we delete all matched vertices together with the incident edges from the current graph in the phase. We repeat an iteration of the outer while-loop until there remains no edge. Then the final M, which collects all matchings obtained during the iteration, gives a maximal matching in G.

Israeli and Shiloach proved that the number of iteration of the outer while-loop is bounded by $O(\log n)$. From their inductive proof, we see that this bound $O(\log n)$ can be replaced by $O(\log \alpha(G))$, where $\alpha(G)$ is the minimum size of a vertex cover in G. Their proof is based on the fact that all vertices of a matching obtained in each phase are contained in a vertex cover in G_i. Moreover, this fact follows from the following two properties: (i) any edge deleted during a phase is incident to at least one active vertex, (ii) if a vertex v becomes active, then v remains active afterwards in the same phase. Since DEGREE-SPLIT contains computing an Euler circuit at step 4 for which $O(\log m)$ time is needed, the time complexity becomes $O(\log \alpha(G) \log \Delta(G) \log m)(= O((\log n)^3))$.

Now, consider the case where a given graph G is bipartite, i.e., G has vertex sets V_1 and V_2 such that any edge in G joins a vertex of V_1 and a vertex of V_2. Then, $\alpha(G) \le \min\{|V_1|, |V_2|\}$, since V_1 and V_2 are both vertex covers in G. Let $\Delta_i = \max\{d(v) \mid v \in V_i\}$ for $i = 1, 2$. We show that the factor $O(\log \Delta(G))$ in the above time bound can be improved to $O(\log \min\{\Delta_1, \Delta_2\})$ by executing the following preprocessing. Assume $\Delta_1 > \Delta_2$ without loss of generality. Consider the set S of vertices of V_1 with degree greater than Δ_2. Before applying DEGREE-SPLIT, we reduce the degree of each vertex v of S to Δ_2 by deleting arbitrary edges incident to v. Since any two vertices of S are not adjacent each other, we can apply this preprocessing to each vertex in S independently. After this preprocessing, every vertex of S will become active at the beginning of the phase. Note that any edge deleted by the preprocessing is indeed incident to an active vertex. The above property (ii) also holds. Hence, this modification does not cause any problem on the analysis of the algorithm. This preprocessing can be done in $O(\log \max\{\Delta_1, \Delta_2\}) = O(\log(|V_1| + |V_2|))$ time. Also, $O(\log m) = O(\log(|V_1| + |V_2|))$. Therefore, we have the following lemma. (Note that $O((\log \min\{n_1, n_2\})(\log \min\{\Delta_1, \Delta_2\})(\log(n_1 + n_2)))$ $= O((\log \min\{\Delta_1, \Delta_2\})(\log n_1)(\log n_2))$.)

Lemma 2. *Let B be a bipartite graph with vertex sets V_1 and V_2. Let $n_i = |V_i|$ and $\Delta_i = \max\{d(v) \mid v \in V_i\}$ for $i = 1, 2$. Then a maximal matching of B can be computed in time $O((\log \min\{\Delta_1, \Delta_2\})(\log n_1)(\log n_2))$ using $O(n_1 + n_2 + \min\{n_1\Delta_1, n_2\Delta_2\})$ processors.* □

Note that in steps 1.3 and 2.3 of JUNGLE, we need to compute a maximal matching in a bipartite graph with $n_1 \le kn$, $n_2 \le m$, $\Delta_1 \le m$ and $\Delta_2 \le k$. From Lemmas 2 and 3, the following theorem is induced. (Precisely, to apply Israeli and Shiloach's algorithm, an input graph needs to be represented by an adjacency list. However, it takes only $O(\log(kn + m))$ time to obtain such data structure, and this time bound is absorbed by the time to compute a maximal matching.)

Theorem 1. *A maximal k-jungle of G can be found in $O((\log n)(\log k)(\log kn)$ $(\log m))$ time using $O(k(n + m))$ processors.* □

Corollary 1. *If G is simple, then a maximal k-jungle of G can be found in $O(\log k(\log n)^3)$ time using $O(k(n + m))$ processors.* □

5 An Efficient Algorithm for a Maximal k-Jungle

In this section, we improve the number of processors required to compute a sparse k-edge-connectivity certificate. We assume that a multigraph G is represented by the simplified graph with the multiplicity function ρ_G. Let m' be the number of edges in the simplified graph $S(G)$.

First of all, we can reduce the multiplicity $\rho_G(e)$ of each multiedge e in a given G to $\min\{k, \rho_G(e)\}$, because a multigraph with the resulting multiplicity is clearly a k-edge-connectivity certificate of G.

Definition 2. *A multigraph obtained from a forest by replacing each single edge with a multiedge of multiplicity r is called a multiforest with multiplicity r.* □

A multiforest with multiplicity r can be seen as r disjoint forests (i.e., r-jungle). Clearly, a multiforest with multiplicity r of a multigraph G with multiplicity $\rho_G(e) \geq r$ (for any multiedge e of G) can be obtained by using $O(n+m')$ processors, because such a multiforest corresponds to a forest in the simplified graph $S(G)$. To treat several multiforests as a jungle, we define a multijungle.

Definition 3. *Let $\mathcal{J} = \{F_1, \ldots, F_h\}$ be an h-jungle. Let $\mathcal{J}'$ be the jungle obtained from $\mathcal{J}$ by replacing each F_i with a multiforest F'_i with multiplicity r_i such that $S(F_i) = S(F'_i)$. Then $\mathcal{J}' = \{F'_1, \ldots, F'_h\}$ is called an h-multijungle with multiplicity $(r_1, \ldots, r_h)$.* □

Note that an h-multijungle with multiplicity $(r_1, \ldots, r_h)$ corresponds to a $(\sum_{1\leq i\leq h} r_i)$-jungle.

We now outline our algorithm for constructing a maximal k-jungle in G. The algorithm consists of $\lceil \log_2 k \rceil$ +1 phases. For simplicity, assume that $k = 2^t$ for some t. In the 0-th phase, we consider the multigraph H_0 induced by the set of edges with multiplicity k, and its simplified graph $S(H_0)$. We then find a maximal forest F_0 in $S(H_0)$ using $O(n+m')$ processors. Note that F_0 can be viewed as a k-jungle $\mathcal{J}(0)$ in H_0 (or a 2-multijungle with multiplicity $(2^{(t-1)}, 2^{(t-1)})$). In the 1-st phase, we augment the k-jungle $\mathcal{J}(0)$ as follows. Regard the $\mathcal{J}(0)$ as a 2-multijungle $\mathcal{J}(0)$, and augment the 2-multijungle using multiedges with multiplicity $\geq 2^{(t-1)}$ in the resulting multigraph, treating each of those multiedges as a single edge. Let $\mathcal{J}(1)$ be the resulting 2-multijungle. In the 2nd phase, we consider the $\mathcal{J}(1)$ as a 4-multijungle with multiplicity $(2^{(t-2)}, 2^{(t-2)}, 2^{(t-2)}, 2^{(t-2)})$, augment $\mathcal{J}(1)$ in the similar manner using multiedges with multiplicity $\geq 2^{(t-2)}$. In the i-phase, given a k-jungle $\mathcal{J}(i-1)$ with form of a $2^{(i-1)}$-multijungle, we augment the $\mathcal{J}(i-1)$ to a k-jungle $\mathcal{J}(i)$ with form of a 2^i-multijungle. By repeating this procedure $t+1$ times, we finally obtain a k-jungle which is maximally augmented.

Now we show the entire description of procedure CERTIFICATE, a phase in our algorithm. For a given integer $k > 0$, we define a vector $(r_1(i), r_2(i), \ldots, r_{2^i}(i))$ for each $i = 0, 1, 2, \ldots, \lceil \log_2 k \rceil$ by $r_1(0) = k$ and recursively setting $r_{2p-1}(i) = \lfloor \frac{r_p(i-1)}{2} \rfloor$ and $r_{2p}(i) = \lceil \frac{r_p(i-1)}{2} \rceil$ for $p \in \{1, 2, \ldots, 2^{(i-1)}\}$. Then it is easy to observe that, for each $i = 0, 1, 2, \ldots, \lceil \log_2 k \rceil$, $\sum_{1\leq p\leq 2^i} r_p(i) = k$, and $\lfloor \frac{k}{2^i} \rfloor \leq r_p(i) \leq \lceil \frac{k}{2^i} \rceil$ for $p = 1, \ldots, 2^i$. For a multijungle $\mathcal{J} = \{F_1, \ldots, F_h\}$, $S(\mathcal{J})$ denotes the set of forests $\{S(F_1), \ldots, S(F_h)\}$. Starting with $i = 0$, $G_{(-1)} = G$ and $\mathcal{J}(-1) = \emptyset$, we repeatedly apply the following procedure for $i = 0, 1, \ldots, \lceil \log_2 k \rceil$.

Procedure CERTIFICATE

Input: a multigraph $G_{(i-1)}$ and a $2^{(i-1)}$-multijungle $\mathcal{J}(i-1) = \{F_1(i-1), \ldots,$

```
        $F_{2^{(i-1)}}(i-1)\}$ with multiplicity $(r_1(i-1), \ldots, r_{2^{(i-1)}}(i-1))$.
Output: a submultigraph $G_i$ of $G_{(i-1)}$ and a $2^i$-multijungle $\mathcal{J}(i) = \{F_1(i), \ldots,$
        $F_{2^i}(i)\}$ with multiplicity $(r_1(i), \ldots, r_{2^i}(i))$.
0. Copy $G_{(i-1)}$ to $G_i$ letting $\rho_{G_i}(e) := \rho_{G_{(i-1)}}(e)$ for all multiedges $e$ of $G_{(i-1)}$.
1. Regard $\mathcal{J}(i-1) = \{F_1(i-1), \ldots, F_{2^{(i-1)}}(i-1)\}$ as a $2^i$-multijungle $\tilde{\mathcal{J}}(i)$
   $= \{F_1(i-1), F_1(i-1), \ldots, F_{2^{(i-1)}}(i-1), F_{2^{(i-1)}}(i-1)\}$ with multiplicity
   $(r_1(i), \ldots, r_{2^i}(i))$ (i.e., each multiforest $F_p(i-1)$ with multiplicity $r_p(i-1)$
   is regarded as two multiforests $F_p(i-1)$ with multiplicity $r_{2p-1}(i)$
   and $r_{2p}(i)$).
2. Augment the $2^i$-multijungle $\tilde{\mathcal{J}}(i)$ to a $2^i$-multijungle $\mathcal{J}(i)$ by adding
   multiedges with multiplicity $\geq \lceil \frac{k}{2^i} \rceil$ in $G_{(i-1)}$ as follows:
   2.1. Let $H_i$ be the submultigraph of $G_{(i-1)}$ induced by multiedges with
        multiplicity $\geq \lceil \frac{k}{2^i} \rceil$.
   2.2. Maximally augment the $2^i$-jungle $S(\tilde{\mathcal{J}}(i))$ using the edges in $S(H_i)$,
        and let $\mathcal{J}'(i) = \{F'_1, \ldots, F'_{2^i}\}$ be the augmented $2^i$-jungle.
   2.3. Let $\mathcal{J}(i) = \{F_1(i), \ldots, F_{2^i}(i)\}$ be the $2^i$-multijungle with multiplicity
        $(r_1(i), \ldots, r_{2^i}(i))$ such that $S(F_p(i)) = F'_p$ for each $p = 1, \ldots, 2^i$
        (such $\mathcal{J}(i)$ can be chosen, since $\rho_{H_i}(e) \geq \lceil \frac{k}{2^i} \rceil \geq r_p(i)$).
3. Update $G_i$ by
   3.1. $\rho_{G_i}(e) := 0$ for all multiedge $e$ of $H_i$ not used by the augmentation, and
   3.2. $\rho_{G_i}(e) := \rho_{G_i}(e) - r_{x(e)}(i)$ for all multiedges $e$ of $H_i$ used by the
        augmentation, where $x(e)$ denotes the $p$ such that $e$ is added to
        $F'_p(i) \in \mathcal{J}'(i)$.
```

After the $\lceil \log_2 k \rceil$-th phase, we obtain a $2^{\lceil \log_2 k \rceil}$-multijungle $\mathcal{J}(\lceil \log_2 k \rceil)$, where the multiplicity of each multiforest in $\mathcal{J}(\lceil \log_2 k \rceil)$ is 0 or 1. By collecting forests multiplicity 1, we obtain a k-jungle. It is easy to show that the multiplicity of any multiedge in G_i is at most $\lceil \frac{k}{2^i} \rceil - 1$. Thus, $G_{\lceil \log k \rceil}$ is an empty graph. We now show that the resulting k-jungle is maximal in G.

Theorem 2. *Let G be a multigraph with $|V| = n$ and $|E| = m$. Let m' be the number of edges in the simplified graph of G. A sparse k-edge-connectivity certificate of G can be found in $O((\log kn)(\log k)^2(\log n)^2)$ time using $O(k(n + m'))$ processors.*

Proof. Let e be a multiedge deleted in the i-th phase, i.e., $\rho_{G_i}(e) := 0$ is executed at step 3.1. Then e is not used by the augmentation of $S(\tilde{\mathcal{J}}(i))$. This means that adding a single edge in e to any forest in the k-jungle $\mathcal{J}(i)$ creates a cycle. For any forest F in the resulting k-jungle, there exists a forest F^* in $\mathcal{J}(i)$ such that $F \supseteq F^*$. Therefore, adding a single edge in e to any forest in the resulting k-jungle also creates a cycle. Any nonjungle edge is deleted in some phase, because $G_{\lceil \log k \rceil}$ is an empty graph. Hence, adding any nonjungle edge to any forest in the resulting k-jungle creates a cycle, i.e., the resulting k-jungle is maximal.

The number of application of CERTIFICATE is $\lceil \log k \rceil + 1$. In each application of CERTIFICATE, all steps except for step 2.2 can be easily computed in $O(1)$ time using $O(k(n+m'))$ processors. At step 2.2, we maximally augment 2^i-jungle $S(\tilde{\mathcal{J}}(i))$ using the edges in $S(H_i)$. Clearly, $|E(S(H_i))| \leq m'$. Thus, applying the algorithm in Section 3, the amount of required resources is $O((\log n)(\log k)(\log kn)(\log m')) = O((\log n)^2(\log k)(\log kn))$ time using $O(k(n+m'))$ processors. (To use JUNGLE, at step 1, we copy the functions D_p, which is used in the previous phase, to D_{2p-1} and D_{2p}.) □

Karger and Motwani gave the following approximate algorithm for minimum cuts which is a parallelization of Matula's $(2+\epsilon)$-approximate min-cut algorithm. Although this algorithm returns an approximate minimum cut size, it is easily modified to find a cut with the returned size.

```
Procedure APPROX-MIN-CUT (multigraph $G$)
1. Let $\delta$ be the minimum degree of $G$.
2. Let $k = \frac{\delta}{2+\epsilon}$.
3. Find a maximal $k$-jungle.
4. Construct $G'$ from $G$ by contracting all nonjungle edges.
5. Return $\min(\delta, \text{APPROX-MIN-CUT}(G'))$.
```

Lemma 3. [5] *Let c be the size of a minimum cut of G. The approximation algorithm returns a value between c and $(2+\epsilon)c$ in $O(\log m)$ levels of recursion.*

Using our algorithm at step 3 of APPROX-MIN-CUT, the following corollary is obtained. Note that $k \leq \delta \leq \frac{2m}{n}$.

Corollary 2. *A $(2+\epsilon)$-approximate minimum cut of G can be found in $\mathcal{NC}$ using $O(\frac{mm'}{n}) = O(nm)$ processors.* □

References

1. J. Cheriyan, M.Y. Kao, and R. Thurimella, Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity, *SIAM J. Comput.* 22 (1993) 157–174. 447
2. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms* (The MIT Press, Cambridge MA 1990). 450
3. A. Israeli and Y. Shiloach, An improved parallel algorithm for maximal matching, *Inform. Process. Lett.* 22 (1986) 57–60. 451
4. J. JáJá, *An Introduction to Parallel Algorithms* (Addison-Wesley, Reading MA 1992). 450
5. D.R. Karger and R. Motwani, An NC algorithm for minimum cuts, *SIAM J. Comput.* 26 (1997) 255–272. 447, 449, 456
6. R.M. Karp and V. Ramachandran, Parallel algorithms for shared memory machines, in: J. van Leeuwen, Ed., *Handbook of Theoretical Computer Science*, Vol. A (Elsevier, Amsterdam 1990) 869–941. 450
7. D.W. Matula, A linear time $2+\epsilon$ approximation algorithm for edge connectivity, in: *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms* (1993) 500–504.

8. H. Nagamochi and T. Ibaraki, A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph, *Algorithmica* 7 (1992) 583–596. 447, 449
9. H. Nagamochi and T. Ibaraki, Computing edge-connectivity of multigraphs and capacitated graphs, *SIAM J. Disc. Math.* 5 (1992) 54–66. 447
10. Y. Shiloach and U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, *J. of Algorithms* 3 (1982) 57–67. 449

A Parallel Algorithm for Sampling Matchings from an Almost Uniform Distribution*

J. Diaz[1], J. Petit[1], P. Psycharis[2], and M. Serna[1]

[1] Departament de Llenguatges i Sistemes
Universitat Politècnica Catalunya, Campus Nord, Mòdul C-6
Jordi Girona Salgado 1–3, 08034-Barcelona, Spain
{diaz,jpetit,mjserna}@lsi.upc.es
[2] Computer Technology Institute (C.T.I.)
Kolokotroni 3, 26221 Patras, Greece
psycharis@cti.gr

Abstract. In this paper we present a randomized parallel algorithm to sample matchings from an almost uniform distribution on the set of matchings of all sizes in a graph. First we prove that the direct NC simulation of the sequential Markov chain technique for this problem is P-complete. Afterwards we present a randomized parallel algorithm for the problem. The technique used is based on the definition of a genetic system that converges to the uniform distribution. The system evolves according to a non-linear equation. Little is known about the convergence of these systems. We can define a non-linear system which converges to a stationary distribution under quite natural conditions. We prove convergence for the system corresponding to the almost uniform sampling of matchings in a graph (up to know the only known convergence for non-linear systems for matchings was matchings on a tree [5]). We give empirical evidence that the system converges faster, in polylogarithmic parallel time.

1 Introduction

The *almost uniform sampling problem* consists in picking at random an element of a finite set according to some distribution Π, such that the variation distance between Π and the uniform distribution is at most ϵ, for a given $\epsilon > 0$. A technique that has proved to be very useful for solving the almost uniform sampling problem, is the *Markov chain technique*. Given a problem, define a Markov chain whose set of states contains all possible solutions, and whose transitions are probabilistic rules that allow a move from state to state. Under certain properties of the underlying graph representing the Markov chain, it can be proved that a polynomial random walk on the states gives an almost randomly generated element from the stationary distribution of the chain, the polynomial is on the size of the input, not on the size of the chain. Over the past years, a large

* This research was partially supported by the ESPRIT LTR Project no. 20244 – ALCOM-IT, CICYT Project TIC97-1475-CE and CIRIT project 1997SGR-00366

body of literature has been devoted to the subject of almost uniform sampling through Markov chains and methods for proving rapid mixing (only a random walk of polynomial length is needed). Excellent surveys can be found in [7,9,2,1] and chapters 6 and 11 of [3].

A question of general interest is the possibility of parallelizing the almost uniform sampling, but the direct NC simulation of the random walk on the underlying Markov chain can be P-complete. For instance, Teng has shown that given one state in the Markov chain used by Jerrum and Sinclair to approximate the Permanent, and a path to compute the state reached by the path is P-complete [8]. However, this result does not exclude the possibility of other kinds of NC samplers.

The problem we are concerned in this paper, consists in, given an undirected graph G, generate in parallel a matching of G, from a distribution as close as desired to the uniform distribution over the set of all matchings in G. Recall that counting the total number of matchings in a given graph is known to be #P-complete [7].

Our parallel sampler is obtained by defining a Genetic System. In such a system, from a given initial distribution, new generations are grown by *mating* two randomly selected parents. Through this paper, the mating operators will produce only one offspring, thus our genetic system is non-quadratic (and non-linear). The generations are new distributions over the set of elements produced by the mating operation. It is worth to remark that through the text, while by a *generation* we mean a distribution over a set, a *population* denotes a multiset produced as a sample of the corresponding generation. In general the convergence of non-linear systems is difficult to prove. The only known results are given in [5] and [4] for symmetric quadratic dynamical systems. Although we could define a quadratic system (instead a genetic one) the resulting system is non-symmetric and therefore the techniques used in [4] cannot be used to show convergence. We establish a close relationship between the evolution of the genetic system an a sequence of Markov chains, and exploit it to show convergence.

Our genetic system for the almost uniform sampling of a matching in a graph can be parallelized for a PRAM. Unfortunately, we have not been able to prove convergence in polylogarithmic time for general graphs. We suspect that the system converges in polylogarithmic time and our experimental results seem to give strong evidence of fast convergence.

The paper is organized as follows. In Section 2, we introduce a Markov chain to generate almost uniformly matchings in a graph and show that a direct simulation of this chain is inherently sequential. To obtain a parallel generator, in Section 3 we define a *genetic system* and prove its convergence to the uniform distribution. Section 4 shows how to implement this system in RNC. Finally, Section 5 gives experimental results in support of our conjecture that the system actually converges in polylogarithmic time.

2 The Markov Chain and Its P-Completeness

Given a graph $G = (V, E)$ with n nodes and μ edges, for any $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, let $M_k(G)$ denote the set of matchings of size k in G, and let $M = \cup_k M_k$ denote the set of all its matchings, with size β.

Let $\mathcal{M}$ be the Markov chain for the uniform sampling of all matchings in a given graph G, defined in [9]. The chain $\mathcal{M}$ contains all the matchings in M as space state, and the transitions are defined by the procedure:

```
function $\mathcal{M}$-Transition ($m \in M$) is
      with Probability 1/2 do
          Sample uniformly an edge $e \in E$
          return Trans($m, e$)
      end with
      return $m$
end

function Trans ($m \in M,\ e \in E$) is
      if $e \in m$ then
          $m := m - \{e\}$
      elsif $m \cup \{e\} \in M$ then
          $m := m \cup \{e\}$
      end if
      return $m$
end
```

The following result is well known (see for example [9]).

Theorem 1. *The Markov chain $\mathcal{M}$ converges to the uniform distribution of all matchings in G and is rapidly mixing.*

Let us show that from a given state in this chain, and a sequence of selected edges, if P≠NC we cannot obtain in NC the final reached state. In formal terms, the previous statement is equivalent to the following problem: Given a graph G, a matching m, a finite sequence of edges $\sigma = (e_1, \dots, e_k)$. Compute the matching m^* obtained by doing a walk from m using σ in $\mathcal{M}$. We shall prove that the above problem is P-complete. To do so, we transform it into a decision problem, add an extra edge e, and rather than asking for m^* we ask whether $e \in m^*$. We call such a problem the *associated problem* for Markov chain $\mathcal{M}$.

Theorem 2. *The associated problem for chain $\mathcal{M}$ is* P*-complete.*

Proof. We present a reduction from the monotone alternating fan out two CVP. Given such a circuit $\alpha = \langle g_1, \dots, g_r \rangle$, we assume that gates are enumerated preserving the layered structure, that means inputs, followed by level 1 gates, followed by level 2 gates, by level 3 gates, ... , followed by the gates at the latest level. Assume that each gate is defined by the equation $g_k = g_i * g_j$ where $*$ represents either AND or OR.

We will construct a bipartite graph, that has two vertices v_i and w_i associated to each gate i, and two additional extra nodes v_0 and w_0. We assume that the graph is the complete bipartite graph, all v_i nodes are in one set and all w_i are in the other.
The initial matching m is the following:

- 1-inputs and OR gates: each node is matched with it's twin, that means we have edges (v_i, w_i).
- 0-inputs and AND gates : each node is unmatched.
- No more edges are added to m.

The sequence is constructed piecewise; a portion from each OR and AND gate are glued together in the gate ordering.
For an OR gate $g_k = g_i$ OR g_j. We define the sequence in three blocks:

- $(w_i, v_0), (w_0, v_j), (w_j, v_0), (w_0, v_i), (w_0, v_j), (w_i, v_0)$
- $(v_k, w_k), (v_i, w_k), (v_k, w_j), (v_k, w_k)$
- $(v_0, w_j), (w_0, v_i), (v_i, w_k), (w_j, w_k)$

The sequence is constructed piecewise; a portion from each OR and AND gate are glued together in the gate ordering.
For an AND gate $g_k = g_i$ AND g_j we add:

- $(w_i, v_0), (w_0, v_j), (w_j, v_0), (w_0, v_i), (w_0, v_j), (w_i, v_0)$
- $(v_i, w_j), (v_j, w_k), (w_i, v_k), (v_k, w_k)$
- $(v_0, w_i), (w_0, v_i), (w_i, v_k), (v_j, w_k), (v_i, w_j)$

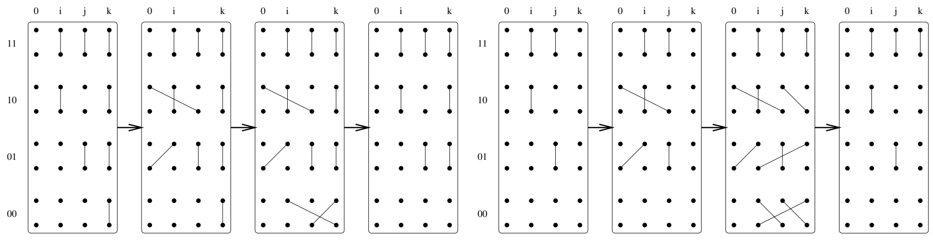


Fig. 1. OR and AND gates before and after processing their corresponding sequences.

The obtained sequence of matchings are sketched in figure 1. The three transitions correspond to the three edge's blocks, in the four possible situations of input values.

It is easy to see that 0's propagate by non-connected nodes and 1's by connected nodes. Therefore, in the final matching we will have an edge joining the nodes associated to the last gate if and only if the circuit outputs true.

One must be careful to understand the statement of Theorem 2. It says that unless NC=P, we can not directly simulate in NC a random walk on the described Markov Chain. It does not say anything about sampling from the almost uniform distribution in NC.

3 The Genetic System and Its Convergence to the Stationary Distribution

In this section we first define a Markov chain associated to any given distribution on the set M, and then present our Genetic system. The evolution of this system can be described as a step on a Markov chain that changes along the time.

Let us consider a probability distribution Π on the set M of all matchings in G. We define a Markov chain $\mathcal{M}(\Pi)$as an extension of the previous chain $\mathcal{M}$: The chain $\mathcal{M}(\Pi)$ has the same state space as $\mathcal{M}$. A transition in $\mathcal{M}(\Pi)$ is defined as follows:

```
function $\mathcal{M}(\Pi)$-Transition ($x \in M$) is
    Sample a matching $y$ according to distribution $\Pi$
    Order randomly the edges of $y$ in a sequence $e_1, \ldots, e_{|y|}$
    $z := x$
    for $i := 1$ to $|y|$ do $z := Trans(z, e_i)$
        return $z$
    end
```

In the particular case that the distribution Π assigns probability $1/2\mu$ to any matchings with exactly one edge, and assigns probability $1/2$ to the empty matching and zero probability to the remaining matchings, the chain $\mathcal{M}(\Pi)$ ccoincides with $\mathcal{M}$.

Let $\Pi(x)$ denote the probability of $x \in M$ under distribution Π. The accessibility of any state in the chain $\mathcal{M}(\Pi)$ can be guarantee in the case that the distribution Π assigns positive probability to all matchings of size 1 in G. In such a case, we can move in $\mathcal{M}(\Pi)$ from any state to any other state in at most n steps. Therefore, the probability of reaching any matching in at most n steps is greater than 0. Thus we get the following lemma.

Lemma 1. *Let Π be a probability distribution such that for every matching x of size 1, we have $\Pi(x) > 0$, then $\mathcal{M}(\Pi)$ is ergodic.*

Given three matchings x, y and z, let $P(x, y, z)$ denote the probability of going from x to z following a sequence given by the edges of y. According with the definition of $\mathcal{M}(\Pi)$ the resulting matching is the same, independently of the order of the edges, thus $P(x, y, z)$ is either 1 or 0. Furthermore $\sum_{z\in M} P(x,y,z) = 1$. The symmetry of $\mathcal{M}$ implies that if we can go from x to z following an ordered sequence of the edges in y, then with the same probability we can go from z to x following the reversed sequence of edges. That is $P(x, y, z) = P(z, x, y)$.

Let us give an expression of the coefficients in the transition matrix of $\mathcal{M}(\Pi)$. Let $\Pi(x, z)$ denote the probability of going from matching x to matching z in one transition of $\mathcal{M}(\Pi)$. According with the definition we have

$$\Pi(x, z) = \sum_{y \in M} P(x, y, z) \cdot \Pi(y).$$

Moreover, as $P(x, y, z) = P(z, y, x)$ we get that $\Pi(x, z) = \Pi(z, x)$. Thus, we obtain the following result.

Corollary 1. *The chain $\mathcal{M}(\Pi)$ is symmetric, for any distribution Π.*

Let us now define a *genetic system* $\mathcal{G}$ over the population of all matchings M. The system will produce the next generation according to a mating rule based in the transitions of $\mathcal{M}(\Pi)$.

Definition 1 (Mating Rule).
From parents x and y, order randomly the edges of y. The offspring z is the matching resulting of the walk in $\mathcal{M}$ starting from state x and following the path defined by the ordered sequence of edges of y.

The system evolves in time $t = 0, 1, \dots$. It starts from a given initial generation Π_0 over M at $t = 0$. The generation at time $t + 1$ is obtained from the generation Π_t at time t, by sampling two matchings x and y according to Π_t, and applying the mating rule to x and y. The system evolves according to the following dynamical equation,

$$\Pi_{t+1}(z) = \sum_{x \in M} \Pi_t(x) \cdot \sum_{y \in M} P(x, y, z) \cdot \Pi_t(y) \tag{1}$$

Theorem 3. *For any distribution Π, the system $\mathcal{G}$ with initial distribution Π and the chain $\mathcal{M}(\Pi)$ have the uniform distribution as fix point.*

Proof. Let us consider the equation

$$\Pi_{t+1}(z) = \sum_{x \in M} \Pi_t(x) \cdot \sum_{y \in M} P(x, y, z) \cdot \Pi_\alpha(y).$$

Notice that with $\Pi_\alpha = \Pi_t$ we have the equation of the genetic system $\mathcal{G}$, and with $\Pi_\alpha = \Pi$ we have the equation of the chain $\mathcal{M}(\Pi)$. Then as $P(x, y, z) = P(z, y, x)$ we get

$$\sum_{x \in M} \Pi_t(x) \cdot \sum_{y \in M} P(x, y, z) \cdot \Pi_\alpha(y) = \sum_{x \in M} \Pi_t(x) \cdot \sum_{y \in M} P(z, y, x) \cdot \Pi_\alpha(y).$$

We must prove that if at some time t we get the uniform distribution $\Pi_t(x) = 1/\beta$ for all $x \in M$, we also get $\Pi_{t+1}(x) = 1/\beta$. Substituting the value of Π_t in the above equation,

$$\begin{aligned}\Pi_{t+1}(k) &= \sum_{x\in M}\frac{1}{\beta}\cdot\sum_{y\in M}P(z,y,x)\cdot\Pi_\alpha(y)\\ &= \frac{1}{\beta}\cdot\sum_{y\in M}\Pi_\alpha(y)\cdot\sum_{x\in M}P(z,y,x)\\ &= \frac{1}{\beta}\cdot\sum_{y\in M}\Pi_\alpha(y) = \frac{1}{\beta}.\end{aligned}$$ □

The previous proof is independent of the choice of the initial distribution. Therefore by Lemma 1, Theorem 3, and classical Markov chain theory we can conclude the following theorem.

Theorem 4. *If for every matching m of size 1 we have $\Pi(m) > 0$, then the chain $\mathcal{M}(\Pi)$ converges to the uniform distribution on the set of all matchings M.*

Let $\Pi_u(x) = \frac{1}{\beta}$ be the uniform distribution on M. Let us recall that obtaining generation $i+1$ in the genetic system $\mathcal{G}$ can be seen as one step in the Markov chain $\mathcal{M}(\Pi_i)$; in other words, assuming $A(\Pi_i)$ is the transition matrix of the chain $\mathcal{M}(\Pi_i)$ then $\Pi_{i+1} = \Pi_i\, A(\Pi_i)$. The last result can be extended to the sequence of Markov chains, provided the initial distribution also assigns nonzero probability to the empty matching.

Theorem 5. *If for every matching m of size 1 or 0, we have $\Pi_0(m) > 0$, then for all $i \geq 0$ the Markov chain $\mathcal{M}(\Pi_i)$ converges to the uniform distribution on the set M.*

Proof. Assume as induction hypothesis that for every matching m of size 1 or 0, we have $\Pi_i(m) > 0$, the condition on the empty matching insures $\Pi_{i+1}(m) > 0$ for any matching m with $\Pi_i(m) > 0$, recall that every matching can be obtained as the mating of itself an the empty matching. As the hypothesis holds for the initial distribution, we get the result.

Let us turn to prove that the genetic system $\mathcal{G}$ also converges to the uniform distribution.

Theorem 6. *If for every matching m of size 1 or 0, we have $\Pi_0(m) > 0$, then the genetic system $\mathcal{G}$ converges to the uniform distribution on the set M.*

Proof. Let us recall that obtaining generation $i+1$ in the genetic system $\mathcal{G}$ can be seen as one step in the Markov chain $\mathcal{M}(\Pi_i)$. The for all i the convergence of $\mathcal{M}(\Pi_i)$ to the uniform distribution for all i. follows by theorem 5.

We first prove that the limit distribution is unique. Assume that Δ is another fix point of $\mathcal{G}$, the restriction that the initial distribution Δ must assign positive probability to any matching implies the Markov chain $\mathcal{M}(\Delta)$ must converge to both Π_u and Δ. As a Markov chain has a unique limit distribution we get $\Pi_u = \Delta$.

Using standard Markov chain techniques (see for example [6]), for every i we get $||\Pi_{i+1} - \Pi_u|| < ||\Pi_i - \Pi_u||$ where $||\cdot||$ denotes the *variation distance* defined as the maximum of the difference of probabilities over all subsets.

This technique can be extended to other mating operations for which the mating verifies the symmetry property $P(x, y, z) = P(z, y, x)$. This is the only property used to shown convergence to the uniform distribution on the set of reachable objects, for all the sequence of chains and the genetic system.

4 Parallel Simulation

In order to simulate the genetic system $\mathcal{G}$ in parallel, we need to implement the following steps in parallel: first to generate a matching according to Π_0, second to compute the mating operation. Moreover, to keep the number of processors feasible, it would be desirable to show that a sample of polynomial size is enough to carry on the simulation.

We select as initial distribution Π_0, the distribution that assigns probability 1/2 to the empty matching and probability $1/2\mu$ to the matchings with one edge. Again notice that for this initial distribution, $\mathcal{M}(\Pi_0)$ and $\mathcal{M}$ coincide. Sampling in RNC from the initial distribution can be done in constant time using as many processors as sample size.

Given two matchings x and y to compute in parallel the mating operation, that gives birth to child z, consider the following procedure:

```
function $\mathcal{M}(\Pi)$-*ParallelTransition* $(x, y \in M)$ is
    for all $e \in x \cap y$ do in parallel $x := x - \{e\};\ y := y - \{e\}$
        for all $e = \{u, v\} \in x$ such that $\exists\{u, w\} \in y$ do in parallel $y := y - \{e\}$
            return $z := x \cup y$
        end
```

Notice that the above procedure can be computed with a PRAM: we represent a matching by two vectors of length n, where the i-th position is either 0, or the vertex matching vertex i, then the resources needed are a constant number of steps and linear number of processors.

Finally, to complete the parallel simulation of $\mathcal{G}$, we have to prove that we don't need an exponential size population to get a good estimate of the system $\mathcal{G}$, otherwise the number of processors needed would be exponential, notice that to generate in parallel a population of exponential size we would need to run in parallel an exponential number of copies of the above procedures. We may overcome this difficulty by showing that a polynomial size population suffices, thus in the overall scheme we will run a polynomial number of copies of processes that use a polynomial number of processors.

We define the following *restricted size* population model: Let s be a parameter to be determined later. At each time t, we maintain a population formed by a multiset F_t of s matchings. At $t = 0$ the initial population is a random s-sample from Π_0. The population F_t at time t, we construct F_{t+1} by executing in parallel and independently s experiments, where each experiment consists of picking uniformly two parents from F_t, selecting who is the left parent with probability 1/2, and apply the mating operator to both matchings to generate a new matching in F_{t+1}. Assume that the experiment finishes at time $t = f$, its value to be determined later.

Using a similar technique as the one described in [4], the discrepancy between this model and the system $\mathcal{G}$ is captured by the concept of a *collision*. We say that a couple of nodes in the same generation *collide* if they share the same father. Given a matching m in F_f, we said that a node in previous populations is *active* if some its edges have been inherited by m. We define a *collision* in the derivation of m if there is a colliding couple such that their common parent is active.

Lemma 2. *The probability that the derivation of an element m in F_f has a collision is bounded by $2n^2f/s$.*

Proof. Recall that we are interested in nodes having edges inherited in m. Hence, at each time t, there are at most n such nodes in F_t. If we fix a level t and consider all derivations such that no pair collides with an active node in levels 1 through t, then the candidates to have edges that end up in m, are selected at random uniformly and independently from $\{1, \ldots, s\}$. The probability of such colliding pair at level t is at most

$$\frac{1}{s}\binom{2n}{2} \leq 2n^2/s,$$

therefore for all t, given that there is no pair that collides with an active node at time less that t, the probability of having a pair that collides with an active node at time t, is at most $2n^2/s$. The lemma follows by summing over all $t \leq f$.

Therefore, for any $\delta > 0$ the restricted size population system and $\mathcal{G}$ remain within variation distance δ for at least f steps provided that the population has size at least $\Theta\left(\frac{n^2 f}{\delta}\right)$. If we wish to have a variation distance smaller than $1/n^2$, a finite population of size $s = \Theta(n^4 p(n))$ suffices to do the simulation, with f a polynomial in n. To finish the simulation, we remove any element from F_f that has a collision in its derivation.

5 Experimental Results

It would be desirable to give a rigorous proof that the non-linear system converges in polylogarithmic number of generations, so we could claim a RNC algorithm for our sampling problem. Nevertheless, we present some empirical evidence in this direction.

In order to test our parallel algorithm and to give an indication that it seems to converge to an almost uniform distribution in polylogarithmic time, we have compared experimentally the two approaches to generate matchings:

Test Graph		Markov Chain	Genetic System	
Name	Nodes	Length	Population Size	Generations
bipartite6	$n = 12$	$n^4 = 20736$	$n^4 = 20736$	$\log_2 n^4 = 15$
random10	$n = 10$	$n^4 = 10000$	$n^4 = 10000$	$\log_2 n^4 = 14$
ring8	$n = 8$	$n^4 = 4096$	$n^4 = 4096$	$\log_2 n^4 = 12$
clique8	$n = 8$	$n^4 = 4096$	$n^5/2 = 16384$	$\log_2^2(n^5/2) = 196$

Table 1. Parameters used for each graph.

the sequential one (via the Markov Chain given in Section 2) and the parallel one (via the Genetic System presented in Section 3). Both techniques were applied to four graph instances in order to generate independently a big number (r) of matchings. The figures showing the frequency of each obtained matching and the frequency of random integers generated using the standard `rand()` function in `stdlib.h` to serve as a control, can be found on the web page: http://www.lsi.upc.es/∼jpetit/Matchings/ParallelSampling. The parameters given to the algorithms to produce these results are shown in Table 1. Remark that we use a polylogarithmic number of steps for all the simulations of the Genetic system.

Comparing the curves of frequencies obtained by Markov Chains and Genetic Systems, we can induce that both techniques generate matchings with almost the same distributions. Moreover, comparing them to the curve obtained using `rand()` we infer that they also follow perfectly a uniform distribution. Only in the random10 graph (a $\mathcal{G}_{n=10,p=1/4}$ graph) we can observe some small discrepancy between the Markov chain and the Genetic system. Due to the considerable density of this graph, a bigger population and a greater number of parallel steps can be required. This was taken into account in the clique8 graph.

Of course, it is arguable that our test instances are rather small and not significative, but this limitation for their size was necessary in order to perform these experiments in a reasonable time (some days). Remember that the Markov chain technique needs $\mathcal{O}(n^4)$ steps and the Genetic system uses n^7 processors.

In order to help to reproduce and verify the measurements and the code mentioned in this research, its code (in C++), instances and raw data are available at

http://www.lsi.upc.es/∼jpetit/Matchings/ParallelSampling

on the World Wide Web.

Acknowledgments

We are thankful to Paul Spirakis for many hours of discussion on the subject.

References

1. M. Jerrum and A. Sinclair. *The Markov Chain Monte Carlo method: An approach to approximate counting and integration*, pages 482–520. PWS, Boston, 1995. 458
2. R. Kannan. Markov chains and polynomial time algorithms. In *35th IEEE Symposium on Foundations of Computer Science*, pages 656–671, 1994. 458
3. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995. 458
4. Y. Rabani, Y. Rabinovich, and A. Sinclair. A computational view of population genetics. In *27th ACM Symposium on Theory of Computing*, pages 83–92, 1995. 458, 465
5. Y. Rabinovich, A. Sinclair, and A. Wigderson. Quadratic dynamical systems. In *33th IEEE Symposium on Foundations of Computer Science*, pages 304–313, 1992. 457, 458
6. A. Renyi. *Probability Theory.* North-Holland, Amsterdam, 1970. 464
7. A. Sinclair. *Algorithm for random generation and counting: A Markov chain approach.* Birkhäuser, Boston, 1993. 458
8. Shang-Hua Teng. Independent sets versus perfect matchings. *Theoretical Computer Science*, pages 1–10, 1995. 458
9. V. Vazirani. Rapidly mixing Markov chains. In B. Bollobas, editor, *Probabilistic combinatorics and its applications*, pages 99–121. American Mathematical Society, 1991. 458, 459

Optimal Approximate Agreement with Omission Faults

Richard Plunkett and Alan Fekete

University of Sydney, Australia
{richard,fekete}@cs.usyd.edu.au

Abstract. Approximate agreement is a building block for fault-tolerant distributed systems. It is a formalisation for the basic operation of choosing a single real value (representing say speed) for use in later computation, reflecting the different approximations to this value reported from a number of possibly-faulty processors or sensors. We develope a new algorithm that can tolerate up to o send-omission faults. We analyse the convergence attained by this algorithm, and also give a matching universal bound on the convergence available to any algorithm no matter how complicated.

1 Introduction

Fault-tolerance is an important property for distributed systems. Distribution makes fault-tolerance possible, since even when some processors fail, there are others which can continue with the computation. Distribution also makes fault-tolerance necessary, since as we increase the number of sites involved in a computation , we increase the likelihood that some will fail during the execution.

There are many different ways in which parts of a distributed system can misbehave during a computation. In general, a fault-tolerant system will be able to produce an appropriate answer in the face of some situations, but others will be so severe that the whole computation is corrupted. To study fault-tolerance more carefully, we describe a system in terms of a *fault model*, which describes the sorts of erroneous behavior that can be overcome. It is clear that more severe forms of behavior will require more sophisticated algorithms to tolerate them. In the theoretical literature, the most studied failure is *byzantine*, which covers a processor changing its state and sending messages arbitrarily. Clearly any algorithm that can overcome such arbitrary faults will be very useful, since almost any imaginable situation will be covered. However, some famous impossibility results [5,8,6] show that byzantine failures are very difficult to overcome. Other work has proposed a model with less severe faults, called *omission faults*, in which a faulty processor may neglect to send some messages, but whenever it does send, the transmitted data is valid. This is an abstraction of the common situation where transmission errors cause identifiable corruption (detected for example by a checksum). It is common in the theoretical literature to present an algorithm that tolerates a bounded number of a certain kind of fault; for example, one might work whenever at most t byzantine errors occur in any execution.

The most famous abstract formulation of a fault-tolerance problem is *agreement* [8], in which each processor has an initial value (representing its state so far), and the processors exchange messages in an attempt to reach a consistent basis for continued computation. It has been shown that solving this, even with very restricted fault types, requires some executions to run for a long time: at least $t+1$ rounds of message exchange.

In order to see what could be to improve response times, Dolev Lynch Pinter Stark and Weihl [2] introduced the problem of approximate agreement, where each processor has an initial real value (say representing a sensor reading), and communication is used to try to give each processor a new value which more accurately reflects the true sensor value. This is formalised by requiring that the final value at any non-faulty processor must be within the range of initial values among the non-faulty processors (the "Validity" condition). The success of such an algorithm is measured by its *convergence*, symbolised by K, defined as the ratio between the spread of final values of non-faulty processors to the spread among initial values of non-faulty processors. This definition means that a small value of K is desirable. Following [2], most work has assumed a single round of message exchange in the algorithm; if extra time is available, one can simply iterate the protocol, reducing the spread of values by a factor of K in each round.

Dealing with Byzantine faults has two major drawbacks, speed and fault tolerance. It is not possible to converge as quickly in the presence of byzantine faults as it may be with lesser faults types and the number of faults that may be tolerated is fewer. Fekete [3] examined the approximate agreement problem for systems with only omission failures. While not providing the variety of fault resistance of byzantine tolerant algorithms Fekete suggests that convergence can be improved by about a factor of about two. Furthermore up to $N-1$ faults may be tolerated in an N processor system, a drastic improvement over byzantine tolerance (for which we must have $N \geq 3t+1$).

We give an algorithm for approximate agreement that works among N processors, whenever there are at most o omission faults. We analyse the convergence of this algorithm, and show that

$$K \leq \left(\left\lceil \frac{N-o}{o} \right\rceil \times 2\right)^{-1}$$

While most of the emphasis in the literature on Approximate Agreement has been on presenting algorithms and analysing their convergence, it is also crucial to understand the theoretical limitations on what any algorithm can do, no matter how cleverly written. Thus we need to find lower bounds on the values of K that are possible for different fault models.

We also prove that every algorithm for this problem (involving a single round of message exchange) must attain convergence that satisfies

$$K \geq \left(\left\lceil \frac{N-o}{o} \right\rceil \times 2\right)^{-1}$$

As can be seen our algorithm exactly matches this bound, proving that it is optimal.

There have been several previous papers in the area of approximate agreement leading to this paper. The problem was defined in [2], which gives an algorithm for systems with synchronous (time-constrained) communication and at most b byzantine failures. This algorithm is found to have performance $K \leq \left(\lceil \frac{N-2b}{b} \rceil\right)^{-1}$ and a matching lower bound is proved. Fekete [3] examines the problem for systems with only omission failures, and gives an algorithm with $K \leq \frac{o}{2N-2o}$; he also proves a lower bound $K \geq \frac{o}{2N+3o}$. Algorithms are found for byzantine faults [2], omission faults [4]. As can be seen convergence can be improved by a factor of two when only omissions faults need to be considered instead of byzantine faults. Azadmanesh and Kieckhafer [7,1] studied MSR (mean-select-reduce) algorithms under mixed-mode (benign, symmetric and asymmetric) faults but presented no general lower bounds. That model didn't really consider omission faults and required they be handled as byzantine. Furthermore the optimal algorithm presented in this paper doesn't fall under the MSR model. The problem has also been considered for systems with asynchronous (unlimited delay) communication. This paper improves on both the algorithm and lower bound by Fekete to a tight result.

The rest of this paper is as follows. In section 2 we present the formal fault model, and also give the important facts about multisets and operations on them, used in the algorithm and its analysis. Section 3 gives the algorithm and the analysis of its convergence rate. Section 4 gives the lower bound. Section 5 concludes.

2 Multisets and Functions and Results

Approximate agreement requires the manipulation of multisets containing real values and the null value($\perp$). A multiset is a collection of values similar to a set, differing in that multiple occurrences of the same value is allowable, the number of times a value occurs in a multiset is referred to as its multiplicity.

There are several representations of these multisets. Let $\Re^+$ be the set of real number plus the null value $\perp$. A multiset may thus be represented as a mapping $V : \Re^+ \rightarrow \aleph$. For each value r from $\Re^+$, $V(r)$ is defined as the multiplicity of r in V, hence $|V| = \sum_{r \in \Re^+} V(r)$.

Another useful representation of the multiset is as a monotonically increasing sequence. So $V = \langle v_0, v_1, \ldots, v_{|V|-1} \rangle$, ordered such that $v_i \leq v_{i+1}$, for convenience of notation we consider the null value to be less than any real value. These two representations are equivalent, though each is best suited to different styles of manipulation. In this work K will be used to mean the convergence rate, and N will be the number of processors in the system.

There are a number of useful functions we need to define on multisets.

- $min(V)$ = the smallest non-null value in V.
- $max(V)$ = the largest non-null value in V.

- $\delta(V) = max(V) - min(V)$ is called the diameter of V.
- $mean(V) = \frac{1}{|V|}\left(\sum_{r\in\Re} V(r)\cdot r\right)$ = the arithmetic mean of the non-null values of V
- Union If $U = V \cup W$ then $\forall r \in \Re^+\ U(r) = max(V(r), W(r))$
- Intersection If $U = V \cap W$ then $\forall r \in \Re^+\ U(r) = min(V(r), W(r))$
- Sum If $U = V + W$ then $\forall r \in \Re^+\ U(r) = V(r) + W(r)$
- Difference If $U = V - W$ then $\forall r \in \Re^+\ U(r) = max(V(r) - W(r), 0)$
- $select_d(v)$ is a subsequence of the multiset V obtained by selecting the first and every subsequent dth element of V. So for $V = \langle v_0, \ldots, v_{|V|-1}\rangle$ we have $select_d(V) = \langle v_o, v_d, v_{2d}, \ldots, v_{jd}\rangle$ where $j = \lfloor \frac{|V|-1}{d} \rfloor$.
- $low_k(M)$ is the multiset formed by taking the k smallest values of M.
- $high_k(M)$ is the multiset formed by taking the k largest values of M.
- $ms_k(M)$ is shorthand for $mean(select_k(M))$

Lemma 1. *if V and W differ by at most t elements where $|V| = |W| = N$ and $V(\perp) = W(\perp) = 0$ then*

$$|mean(select_t(V)) - mean(select_t(W))| \leq \delta(V \cup W) \cdot \left(\left\lceil \frac{N-t}{t} \right\rceil\right)^{-1}$$

This is a standard result in the field [2,7].

3 The Algorithm

The algorithm has each processor start with a value. That value is then sent to each other node in the network. Each node collects all the values it receives and places them into a multiset. For the purposes of this algorithm absent values, null values and other erroneous values (eg values that fail the checksum) are ignored, leaving between N and $N - o$ real values in the multiset received at each node.

1. Collect multiset M of N to $N - o$ real values, one from each processor, ignoring nulls.
2. The new approximation for this node is

$$approx(M) = (ms_o(low_{N-o}(M)) + ms_o(high_{N-o}(M)))/2$$

To clarify we present an example of applying this function. Let $N = 7$ and $o = 2$. Consider a node which receives $V = \langle 0, 1, 1, 2, 2, 2\rangle$ for its multiset (note one value is missing). $low_{N-o}(V) = \langle 0, 1, 1, 2, 2\rangle$ and $high_{N-o}(V) = \langle 1, 1, 2, 2, 2\rangle$. So $mean(select_o(low_{N-o}(V))) = mean(\langle 0, 1, 2\rangle) = 1$ and $mean(select_o(high_{N-o}(V))) = mean(\langle 1, 2, 2\rangle) = \frac{5}{3}$. So $approx(V) = \frac{4}{3}$.

3.1 Analysis of Convergence

We will consider an arbitrary execution and analyse the maximum difference that can occur between the approximations as calculated by any two nodes. Take a set of $N-o$ processors from the non-faulty processors in the execution, let V_{nf} be the multiset of values held at these processors. Clearly V_{nf} is a subset of the values received at each node during the execution. Let V_{all} be the multiset of all values held at each node initially. Thus V_{all} is a superset of the values received at each node.

From the approximation function we can observe that $approx(V_{nf}) = ms_o(V_{nf})$. The way the algorithm works is for each processor to calculate the highest and lowest values that $ms_o(V_{nf})$ can take that would be consistent with the multiset it received, and average those estimates to form its own new approximation.

Let $V_l = low_{N-o}(V_{all})$ and $V_h = high_{N-o}(V_{all})$, Thus for a processor which received V_{all} the new approximation would be $(ms_o(V_l) + ms_o(V_h))/2$. Consider bound on the value $ms_o(V_h) - ms_o(V_l)$. V_l and V_h have at most o differences, thus from lemma 1 the maximum difference between the values $ms_o(V_h)$ and $ms_o(V_l)$ is given by $ms_o(V_h) - ms_o(V_l) \leq \delta(V_{all}) \cdot \left(\left\lceil \frac{N-o}{o} \right\rceil\right)^{-1}$.

Lemma 2. *If $|M_1| \geq N-o$ and $|M_2| \geq N-o$ and $M_2 \subseteq M_1$ then $ms_o(low_{N-o}(M_1)) \leq ms_o(low_{N-o}(M_2))$ and $ms_o(high_{N-o}(M_1)) \geq ms_o(high_{N-o}(M_2))$.*

Proof. M_2 is a subset of M_1, thus we can be assured that every one of the $N-o$ smallest elements of M_2 appears in M_1. Thus the ith smallest element of M_1 is less than of equal to the ith smallest element of M_2. Consequently the ith smallest element of $low_{N-o}(M_1)$ is less than or equal to the ith smallest element of $low_{N-o}(M_2)$. Since $ms_o()$ selects out the same index values from $low_{N-o}(M_1)$ and $low_{N-o}(M_2)$ and then averages them we know that the values from M_1 will be less than or equal to the values selected from M_2 Thus the average of the values from M_1 will be less than or equal to average of the values selected from M_2, so $ms_o(low_{N-o}(M_1)) \leq ms_o(low_{N-o}(M_2))$. By a symettrical argument $ms_o(high_{N-o}(M_1)) \geq ms_o(high_{N-o}(M_2))$

A direct result of this is that for any multiset Z received at some processor during the execution we have $ms_o(low_{N-o}(V_{all})) \leq ms_o(low_{N-o}(Z)) \leq ms_o(V_{nf}) \leq ms_o(high_{N-o}(Z)) \leq ms_o(high_{N-o}(V_{all}))$.

Consider now two multisets X and Y received by two separate processors during the execution. Assume wlog that $approx(X) \leq approx(Y)$. From the above we know that $ms_o(low_{N-o}(V_{all})) \leq ms_o(low_{N-o}(X))$ and $ms_o(V_{nf}) \leq ms_o(high_{N-o}(X))$ and $ms_o(low_{N-o}(Y)) \leq ms_o(V_{nf})$ and $ms_o(high_{N-o}(Y)) \leq ms_o(high_{N-o}(V_{all}))$. So

$$\begin{aligned}
approx(X) &= (ms_o(low_{N-o}(X)) + ms_o(high_{N-o}(X)))/2 \\
&\geq (ms_o(low_{N-o}(V_{all})) + ms_o(V_{nf}))/2 \\
approx(Y) &= (ms_o(low_{N-o}(Y)) + ms_o(high_{N-o}(Y)))/2 \\
&\leq (ms_o(V_{nf}) + ms_o(high_{N-o}(V_{all})))/2 \\
approx(Y) - approx(X) &\leq (ms_o(V_{nf}) + ms_o(high_{N-o}(V_{all})))/2 \\
&\quad - (ms_o(low_{N-o}(V_{all})) + ms_o(V_{nf}))/2 \\
&\leq ms_o(high_{N-o}(V_{all}))/2 - ms_o(low_{N-o}(V_{all}))/2 \\
&\leq \delta(V_{all}). \left(\left\lceil \tfrac{N-o}{o} \right\rceil\right)^{-1}/2 \\
K &\leq \left(\left\lceil \tfrac{N-o}{o} \right\rceil \times 2\right)^{-1}
\end{aligned}$$

Which gives the convergence for one round of the algorithm.

4 Lower Bound on Convergence of Arbitrary Algorithm

Any algorithm can be represented as a full information protocol, in which values are exchanged and then some protocol-specific function is applied to the collected messages to determine the final value. Thus we define a view as the vector of values received at each processor after the communication step. It contains a list of values and the processors they came from. Typically only the values are used, the vector being converted immediately into a multiset. Consider an execution E in this fault model. It may be described by the starting value of each node and stating which nodes are faulty, and how those faulty nodes behave. During E not all nodes need receive the same data vector (view) due to faulty behaviour of some nodes as specified in E. We say that two views V and W are directly compatible ($V \approx W$) if there can exist an execution E in which V can be received at one correct processor and W be received at another correct processor. In this situation we say that E produces V and W ($E \Rightarrow (V, W)$).

Note that any approximation function f run at a node must produce a value in the range of values received at that node.

In our lower bound we will use a simple counting argument, expressed in the following result.

Lemma 3. *Let f be any valid approximation function. Suppose that V_0,V_1,...,V_c is a chain of views such that $V_i \approx V_{i+1}$ and where $f(V_0) = 0$ and $f(V_c) = 1$. There exists an i $(0 \leq i < c)$ such that*

$$|f(V_i) - f(V_{i+1})| \geq \frac{1}{c}$$

Proof. We argue by contradiction, and let $S = \langle s_1, s_2, \ldots, s_c \rangle$ where $s_i = f(V_i) - f(V_{i-1})$.

$$\sum_{i=1}^{c} s_i = (f(V_1) - f(V_0)) + (f(V_2) - f(V_1)) + \cdots + (f(V_c) - f(V_{c-1})) = f(V_c) - f(V_0) = 1$$

However, if $s_i < \frac{1}{c}$ for all i then $\sum_{i=1}^{c} s_i < c * \frac{1}{c}$ so $\sum_{i=1}^{c} s_i < 1$, which, since $\sum_{i=1}^{c} s_i = 1$, isn't true. So for some i $s_i \geq \frac{1}{c}$

We can now use the lemma to establish our lower bound.

Theorem 1. *If A is any algorithm that solves Approximate Agreement in all executions of a distributed systems with at most o omission failures, then the convergence of A is bounded below by*

$$K \geq \left(2 \times \left\lceil \frac{N-o}{o} \right\rceil \right)^{-1}$$

Proof. We will produce a chain of views of length c, such that every consecutive pair are directly compatible, the approximation function value produced from the first must be 0, the approximation function value produced from the last must be 1, and every one contains only entries of 0, 1 or $\perp$. The lemma above will then show that one execution exists in which two neighbouring views are produced, where the final values differ by at least $\frac{1}{c}$. In this execution, all initial values are 0 or 1, showing that the interval of values is reduced to no less than $1/c$; that is, the convergence is no less than $1/c$.

We will now complete the lower bound by providing a specification of a chain of executions $E_0, \ldots, E_{c-1}$ which produces a chain of views $V_0, \ldots, V_c$ ($E_i \Rightarrow (V_i, V_{i+1})$) which satisfies the above conditions. The precise characterisation is given below, but we offer a diagram that may make things clear more simply. The chain starts with all entries being 0 except for some nulls, whose total number is less than the maximum faults; thus the decision value must be 0. Similarly the decision value in the final view must be 1, with all entries (except a few nulls that might be from faulty processors) being 1. In between views in the chain are made by gradually replacing zero-value entries with nulls and 1-value entries, as shown in Figure 1.

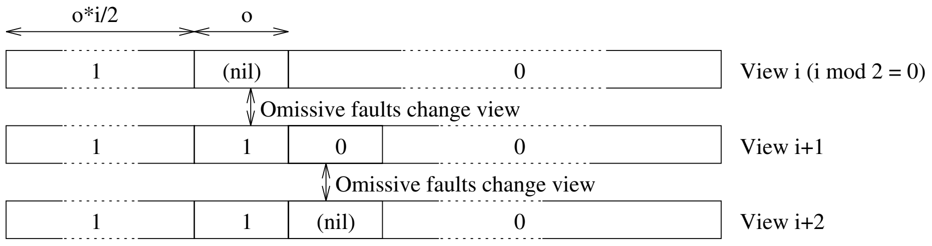


Fig. 1. Construction of the chain of views

Now to make this precise, we indicate exactly what happens in each execution, giving different constrictions for alternating pairs in the chain. In the execution E_i containing nodes $\langle p_0, ..., p_{N-1}\rangle$ where $i \bmod 2 = 0$ the nodes behave as follows. (All ranges are inclusive.)

– Nodes p_0 to $p_{(\frac{i}{2}+1)*o-1}$ start with value 1.

- The remaining nodes all start with value 0.

In the message exchange, the following is observed

- Nodes $p_{\frac{i}{2}*o}$ to $p_{(\frac{i}{2}+1)*o-1}$ are omissively faulty, they contribute $\perp$ to view V_i and 1 to the view V_{i+1}.
- The remaining nodes are non-faulty

In execution E_{i+1} where $i \bmod 2 = 0$ the nodes behave as follows.

- Nodes p_0 to $p_{(\frac{i}{2}+1)*o-1}$ start with value 1.
- The remaining nodes all start with value 0.

The behavior in these executions is as follows

- Nodes $p_{(\frac{i}{2}+1)*o}$ to $p_{(\frac{i}{2}+2)*o-1}$ are omissively faulty, they contribute 0 to view V_{i+1} and $\perp$ to the view V_{i+2}.
- The remaining nodes are non-faulty

As can be seen V_0 contains only values of 0 and $\perp$, so that and valid f will have $f(V_0) = 0$. E_i and E_{i+1} both produce the same thing for V_{i+1}, so for any V_i and V_{i+1} we have $V_i \approx V_{i+1}$ because $E_i \Rightarrow (V_i, V_{i+1})$. We now need only show $f(V_c) = 1$ for any f.

Notice that the number of 0's in each view is reducing by o every two steps.($V_{i+2}(0) = V_i(0) - o$ or $V_{i+2}(0) = 0$). We set $c = \lceil \frac{N-o}{o} \rceil \times 2$. This is sufficient to ensure $V_c(1) \geq N - o$. Since the only other value in V_c is $\perp$ it follows that $f(V_c) = 1$ for any valid f.

This completes the construction of the chain, and thus by Lemma 3 we have for any approximation function f the convergence rate is bound by

$$K \geq \left(\left\lceil \frac{N-o}{o} \right\rceil \times 2 \right)^{-1}$$

5 Conclusion

We have analysed the Approximate Agreement problem for distributed systems subject to at most o omission failures. We have presented an algorithm and shown its convergence to be

$$K \leq \left(\left\lceil \frac{N-o}{o} \right\rceil \times 2 \right)^{-1}$$

We have also proved a matching lower bound thus guaranteeing that the algorithm presented is optimal for this faults model. Both algorithm and lower bound improving on those presented in previous works.

In future work we plan to extend our analysis to hybrid faults systems and to the closely related task of clock synchronisation.

References

1. M. Azadmanesh, R. Kieckhafer, “New Hybrid Fault Models for Asynchronous Approximate Agreement” *IEEE Trans on Computers*, **45**(4):439–449, (1996). 469
2. D. Dolev, N. Lynch, S. Pinter, E. Stark, W. Weihl, “Reaching Approximate Agreement in the Presence of Faults”, *Journal of the ACM*, **33**(3):499–516, (1986). 468, 469, 470
3. A. Fekete, “Asymptotically Optimal Algorithms for Approximate Agreement”, *Distributed Computing*, **4**:9–29, (1990). 468, 469
4. A. Fekete, “Asynchronous Approximate Agreement” *Information and Computation* **115**(1):95–124, (1994). 469
5. M. Fischer, N. Lynch, “A Lower Bound for the Time to Assure Interactive Consistency”, *Information Processing Letters* **14**(4):183–186 (1982). 467
6. M. Fischer, N. Lynch, M. Patterson, “Impossibility of Distributed Consensus with One Faulty Process”, *Journal of the ACM*, **32**(2):374–382 (1985). 467
7. R. Kieckhafer, M. Azadmanesh, “Reaching Approximate Agreement with Mixed-Mode Faults”, *IEEE Trans on Parallel and Distributed Systems*, **5**(1):53–63, (1994). 469, 470
8. M. Pease, R. Shostak, L. Lamport, “Reaching Agreement in the Presence of Faults”, *Journal of the ACM* **27**(2):228–234 (1980). 467, 468

Author Index